App Builder CET W/32
User guide

Index

Chapter 1 Installation guide 9

1.1 Introduction. 9

1.2 Requeriments. 9

1.3 Distribution software 9

1.4 Installation of Application Builder CET W/32 9

1.5 Windows Environment variables 11

1.6 Variables de entorno de Cet 11

1.7 Key protection activation 15

1.8 MACADDRESS CET Protection 15

1.9 Verifying the Installation under Windows 16

1.10 Web site 16

1.11 Versions 16

Chapter 2 Overview 17

2.1 Introduction 17

2.2 The base language 17

2.3 The W/32 environment. 17

2.4 The W/32 IDE. 18

2.5 Subroutines BASIC 19

2.6 Mouse support 19

2.7 Viewing graphic images. 19

2.8 ODBC support 19

2.9 MySQL 20

2.9.1 Introduction 20
2.9.2 New features 20
2.9.3 Non supported features 20
2.9.4 References 20
2.9.5 Working with MySQL data base. 20
2.9.6 Files indexed emulation. 20
2.9.7 Configuración 21
2.9.8 File access methods and formats 21
2.9.9 File names 22
2.9.10 Files and block records 22
2.9.11 Natives APIs 22
2.9.12 Example of Using native APIs CET BASIC programs. 25
2.9.13 Installing and configuring the MySQL server 27
2.9.14 Configuration File Access emulator 28
2.9.15 Moving ISAM data files to MySQL tables 29
2.9.16 Example 30
2.9.17 Creating ISAM file to conversion 30
2.9.18 Analyzing the isam file 31
2.9.19 Exporting indexed ISAM data files to text files 32
2.9.20 Creating MySQL tables and data imports 33
2.9.21 Modifing CET BASIC programs 33

2.10 CET W/32 Library 33

2.10.1 The POS windows library 33

2.10.2 The bar code CET library 34
2.10.3 The file access library. 34
2.10.4 Future versions 35

Chapter 3 The W/32 Compiler 36

3.1 Introduction. *36*
3.2 The W/32 components. *36*
3.3 Creating W/32 programs *36*
3.4 Compiling a program with W/32 *36*
3.5 The compilations 'flags' *37*
3.5.1 32 bits Option (-32) 37
3.5.2 Generate assembly code as output (-S) 37
3.5.3 Define a symbol (-D*símbolo*) 37
3.5.4 Select line error (-#) 37
3.5.5 Float text editor (-Zi) 38
3.5.6 Define the directory path headers. H (-I*path*) 38
3.5.7 Include a library on linker (-l*name*) 38
3.5.8 Line numering (-Ln) 38
3.5.9 Only compile .obj (-c) 39
3.5.10 Especify the .exe file name (-o nom_ejecutable) 39
3.5.11 Preprocessor and preprocessor only (-P y -PO) 39
3.5.12 Report (-R*paso*) 39
3.5.13 Resource (-r nombre) 39
3.5.14 Track code -trace- (-T[*rastrea-sub*]) 39
3.5.15 Ux-basic (-U) 40
3.5.16 Version number opction (-V) 40
3.5.17 Windows Objects (-wo nombre) 40
3.5.18 Windows resources (-wr nombre) 40
3.6 Environment variables used by the compiler *40*
3.6.1 B_CMDFLAGS 40
3.6.2 B_THCONV 40
3.7 Environment variables used by the runtime system *41*
3.7.1 Variables related to the compiler and runtime system 41
3.7.2 B_CETBIN 41
3.7.3 B_CETINC 41
3.7.4 B_CETLIB 41
3.7.5 B_BRTMDIR 41
3.7.6 Environment Variables related files and input / output 41
3.7.7 B_CSHFCLOSE 41
3.7.8 B_FPATH 41
3.7.9 B_LIBFNTYP 41
3.7.10 B_LKPATH 42
3.7.11 B_REOPENS 42
3.7.12 Variables affecting the use and control of printing 42
3.7.13 B_FFPTR 42
3.7.14 B_MAPPRTn 42
3.7.15 B_PRINTERn 42
3.7.16 B_WINPRINT 42
3.7.17 Variables affecting the screen 42
3.7.18 B_DFLTBGC 42
3.7.19 B_DFLTFGC 43
3.7.20 B_SCRNOLINES 43
3.7.21 Variables that provide support THEOS 43
3.7.22 B_<código-driver>DRIVE 43
3.7.23 B_DOSTAB 43
3.7.24 B_EMULATE 43

3.7.25 B_OPTLOCK 43
3.7.26 B_THLOCK 43
3.7.27 B_THPON 44
3.7.28 B_THVAL 44
3.7.29 Generic Environment Variables 44
3.7.30 B_4DYEAR 44
3.7.31 B_DATEFORM 44
3.7.32 B_LANGUAGE 44
3.7.33 B_USER 44
3.7.34 B_ODBCERR 44
3.7.35 B_YR2000 44

Chapter 4 The framework 45

4.1 Introduction 45

4.2 The default windows. 45
4.2.1 The file menu 45
4.2.2 The View menu 45
4.2.3 The Font menu 45
4.2.4 The Help menu 46

4.3 The framework default windows 46

4.4 Changing the default framework 46

4.5 Selecting fonts 47
4.5.1 Delete the Font menu 47
4.5.2 Choosing the sources within the Windows Administrator 47
4.5.3 Displaying the default toolbar 48
4.5.4 Viewing the status bar by default 48
4.5.5 Changing the default title in the main window 48
4.5.6 Changing the default menu bar 48
4.5.7 Adding specific messages for menu items 50
4.5.8 Changing the message About 50
4.5.9 Changing the title of a message box w/32 50
4.5.10 Changing the program icon 51
4.5.11 Simulating keyboard input with mouse subroutines. 51
4.5.12 Special considerations when using windows 53

Chapter 5 The W/32 integrated develop environment 54

5.1 Introduction 54

5.2 Use Project - Basics 54

5.3 The program W32APBLD 54
5.3.1 The Help menu 55
5.3.2 The View menu 55
5.3.3 The File menu 55
5.3.4 Command Arguments W32APBLD 55

5.4 Create a project 55

5.5 Adding files to project 56

5.6 Building and maintaining a project 57

5.7 Create subprojects 58

5.8 W32 examples Projects 59

5.9 Rename, Copy or Move existing projects. 59
5.9.1 Rename a project 59
5.9.2 Move or copy a project. 59

Index

Chapter 6 The debugger W/32 60

6.1 Introduction 60

6.2 Using the Source Code Editor 60

6.3 Using the debugger W/32 60
6.3.1 Enable debugger. 60

Chapter 7 The dialog editor W/32 62

7.1 Introduction 62

7.2 Creating a user interface 62

7.3 The dialog editor w/32 63
7.3.1 Moving a control 63
7.3.2 Resizing a Control 63
7.3.3 Copying a Control 63
7.3.4 Deleting a Control 64
7.3.5 Selecting multiple controls 64
7.3.6 Undo and Redo options 64

7.4 Opening an existing dialog 64

7.5 Save a dialog 64

7.6 Utility: Dialog templates. 64

7.7 Defining a dialogue or a display format 65
7.7.1 Tittle 66
7.7.2 Prefix 66
7.7.3 Type 66
7.7.4 Size 66
7.7.5 Source file 66

7.8 Defining controls 67
7.8.1 Check Box 68
7.8.2 Combobox 69
7.8.3 Command buttons 70
7.8.4 ContEdit text 70
7.8.5 Frames 71
7.8.6 Listboxes 71
7.8.7 Radio buttons 72
7.8.8 Scrollbars 73
7.8.9 Labels 74

7.9 Changing the look and how to operate the dialog 74
7.9.1 Alignment 74
7.9.2 Centered 75
7.9.3 Using the Same Size 75
7.9.4 Tab order 75
7.9.5 Returning to create a sequence of IDs 75
7.9.6 Screen Resolution 76
7.9.7 Radio buttons commands 76
7.9.8 Grid Size (Resolution) 76

7.10 Runtime functions 76
7.10.1 Checking Box Controls 76
7.10.2 Combo box controls 76
7.10.3 Command buttons controls 77
7.10.4 Edit controls 77
7.10.5 List box controls 77
7.10.6 Radio buttons controls 78
7.10.7 Scrollbars controls 78

7.10.8 General routines 78

7.11 Dialog source files 79
7.11.1 Program upper part 80
7.11.2 Prefix subroutine 81
7.11.3 prefixBtnClick subroutine 81
7.11.4 PrefixEvent subroutine 83
7.11.5 PrefixInit subroutine 84

7.12 Copying a dialog 84

7.13 The main program 84

7.14 INVOICE - An example using visual format 86
7.14.1 The main program 87
7.14.2 The visual format source file 87
7.14.3 Dialog source files 93

7.15 VALIDATE - Data validation subroutine 96

7.16 W32 DEMOS - A menu example program 98

7.17 CALC - A special buttons proccesing program 100

7.18 PROGRESS - A dialog example with progress bar 101

Chapter 8 W/32 Window Functions 103

8.1 Introduction 103

8.2 Using C language routines 103
8.2.1 Functions list 117

8.3 W/32 functions list 118

Chapter 9 BASIC subroutines 122

9.1 Introduction 122

9.2 Compiling programs that call BASIC Subroutines 122

9.3 BASIC subroutines 122

9.4 $MAIN sentence 122

9.5 $SUB sentence 123

9.6 $GLOBAL sentence 123

9.7 $CLEAR sentence 124

9.8 $EXIT sentence 124

9.9 Converting existing applications in BASIC CET 124

9.10 A conversion example of an application BASIC CET 125

9.11 Differences cet basic conventional programs 125
9.11.1 Error handling 126
9.11.2 RESTORE y READ with the DATA sentences 126
9.11.3 COMMON sentences 126
9.11.4 User defined functions 126

Chapter 10 ODBC support 127

10.1 Introduction 127

10.2 Opening a file 127

10.3 Reading a file 128

Index

10.4 Updating a file 129
10.4.1 Updating a column 129
10.4.2 Deleting a record 129

10.5 An example program 130

Chapter 11 CALLS FROM WIN32 API CET W32. 132

Chapter 12 Development Utilities 202

12.1 Introduction 202

12.2 Bcopy 202

12.3 Bcreate command 203

12.4 El comando Blist 203

12.5 Bpretty command 204

12.6 Brenum command 204

12.7 CETLNK32 command 205

12.8 Dcheck command 205

12.9 BSORT command 206
12.9.1 Instalation: 207
12.9.2 BSORT sintax 207

Chapter 13 Routines performed in C 216

13.1 Introduction. 216

13.2 Routines performed in C 216
13.2.1 Functions of files 216
13.2.2 Bcreate 216
13.2.3 Berase 216
13.2.4 Brename 216
13.2.5 Bcopy 216
13.2.6 Bfilestat 217
13.2.7 Bgetcwd y Bchdir 217
13.2.8 Bindinfo 217
13.2.9 Functions related to the environment 217
13.2.10 Bgetenv y Bputenv 217
13.2.11 Bgetenv32 y Bputenv32 217
13.2.12 Bmemavl y Bmemmax 217
13.2.13 Bsernum 218
13.2.14 Functions of Program 218
13.2.15 Becho y Bnoecho 218
13.2.16 Bkdshift 218

Chapter 14 Implementation Notes. 219

14.1 Introduction 219

14.2 El Default PIF file 219

14.3 Current working directory 219

14.4 The name of a file 219

14.5 The default search sequence 220

14.6 The number of open files 220

14.7 CSI/CSH sentences 220

14.8 Msgbox Sentence _____ *221*

*14.9 Printing*_____ *221*

14.9.1 Native printing _____ 221

14.9.2 Printing text in Windows printers not compatible with MS-Dos _____ 222

14.9.3 GRAPHICS PRINTING WITH WINDOWS SOURCES _____ 222

Chapter 1 Installation guide

1.1 Introduction.

The Application Development System w/32 CET is a compiler, libraries and utilities designed to produce commercial applications in 32-bit Windows operating systems from Microsoft.
This chapter will provide the information necessary to install software w/32 of CET.

1.2 Requeriments.

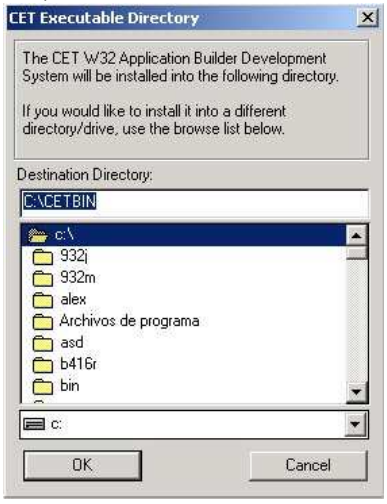
The Application Development System w/32 CET is available both as Development System RunTime.
Development: the platform is recommended for Windows 9x, 2000, XP or NT running on a 486 or higher, with at least 168 MB of RAM. It takes about 32 MB free hard drive space.
For RunTime system, we recommend Windows 9x, 2000, XP or NT running on a PC 386 or higher with at least 8 MB of RAM.

1.3 Distribution software

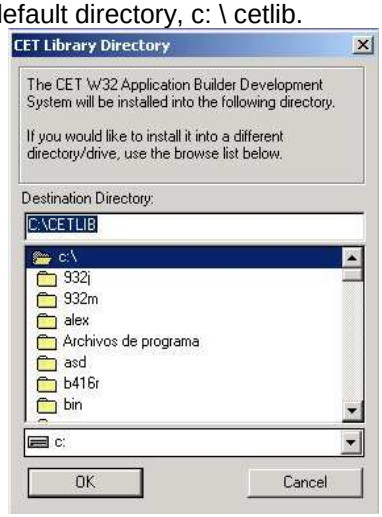
All is avaliable at http://www.omegasystems.com and http://www.cet-software.com/

1.4 Installation of Application Builder CET W/32

To install the Application Development System w/32 CET, enter the basic Cet CD in the CD-ROM, open the folder "cet932u Installer" and run "Setup.exe"
The setup program will ask the directory to use for storing executable w/32 CET. We recommend that you accept the default directory that is shown, c: \ cetbin.



The SETUP program also provides a directory that will be used to store components w/32 Seller CET. We recommend using the default directory, c: \ cetlib.



Chapter 1 Installation guide

This document is a tool for developers who want to build an installer for your software, incorporating the necessary files CET BASIC as part of the installation. Thus, the developer could make a single installer for their own software, without the customer (or the developer himself) have to make the process of installing the software on each machine CET runtime separately. This document was written for version 9.31a w/32, but has notes on other versions.

I. CETBIN

The CET runtime installer sets by default C: \ CETBIN as target directory, but should be able to run on almost any other directory. Most parts of the runtime CET is housed in that directory. Developer may want to install the software application in the same directory as CET runtime components, thus avoiding the need to create a new directory on the user's computer.

bcheck.exe
bcopy.exe
bcreate.exe
blist.exe
brun.err
butil.err
butil.err
butil.msg
cbeprint.dll (added on version 9.30)
cetauth.exe
cetkprnt.exe
cetprnt1.bat
cetrt.dll
dcheck.exe
runw32.exe

II. WINDOWS

The CET runtime installer does not install anything in the WINDOWS directory. However, you must install a copy of the file there or <Application> W32APBLD.w32. W32, since the application will need this file to operate properly. CET not installed with the runtime, since the contents of the file will depend on the developer's application. Perhaps you want the copy the installation software does not exist, or overwrite it under certain conditions. Or, maybe you just want to install it in new installations, and ignonarlo on upgrades.
W32APBLD.w32
-o- <aplicacion>.w32

III. WINDOWS\SYSTEM

The CET runtime installer copies the following files to the C: \ WINDOWS \ SYSTEM.

imagehlp.dll
mfc42.dll
msvcirt.dll
msvcrt.dll
msvcrt40.dll
netbios.dll
odbc32.dll
odbcint.dll

When the compiler is installed in a directory other than CETBIN, if using a shortcut to the main application program without preceding it with 'drive: \ <directory> \ runw32.exe', be sure to set the value of the variable environment 'B_CETBIN' in the *. w32 appropriate (either <Application>. w32 or W32APBLD.w32), with the directory where you saved the runtime. Otherwise can not locate the runtime files, which may result in an error or incorrect operation.
WINDOWS directory may have other names (eg. WIN95, WINNT, or other user-defined). You will need to obtain the name of the directory the installation process..

1.5 Windows Environment variables

At the end of the installation, the program will ask you how to adjust your autoexec.bat to ensure your w/32 CET function properly.
During compilation, the Application Development System w/32 CET depends on the contents of the environment variables PATH, LIB and INCLUDE to determine where the components of the system. In these variables must enter specific directories during the installation process. For example, if you used the recommended directories by default, update your autoexec.bat adding c: \ cetbin defining the existing PATH and LIB and INCLUDE variables INCLUDE directory should be loaded as drive: \ <PATH-lib> \ INC, where <PATH-lib> is the same as the LIB directory.
. SET PATH = C: \ CETBIN
SET LIB = C: \ CETLIB
SET INCLUDE = C: \ CETLIB \ INC
If you have a copy of DOS product in this system, make sure that it is the directory containing the files of MASM, DOS used by the product, rather than the \ cetbin in the list of directories in the PATH variable. Otherwise, the assembly will fail when you run your program.
See section Environment Variables Compilation and Execution in Chapter w/32 Compiler for more information on how to load other variables necessary to run your application.
For versions of Windows based on NT technology, the environment variables are modified in the System window, Advanced and clicking the button "Environment Variables", all from the Control Panel:
Path variable is amended by adding the directory in which you installed the CETBIN folder as the model "C: \ CETBIN"
Variables are added SET LIB = C: \ CETLIB and SET INCLUDE = C: \ CETLIB \ INC

1.6 Variables de entorno de Cet

B_CMDFLAGS
Fixed compiler options making them available for each build. For example, if you use the library (optional) Phase One Toolkit, you should enter the library option to tell the compiler where to find these routines.
B_CMDFLAGS = "-lPostk"
The following example illustrates how you can select B_CMDFLAGS OBW32 so that files always use the Framework shortlisted located in the current directory. Separate multiple options with a space like this:
B_CMDFLAGS = "-r mires" "-wo miobj"
This variable will tell the compiler to enable the defaults and display their values. Selecting this variable is no longer needed in the command line options present in the variable.
B_THCONV
Loads 1 to convert string literals to values THEOS. This will ensure that the codes can be converted to international characters (ASCII values above 128) to your assembly output.
B_THCONV = 1
Environment variables used by the runtime system
A lot of variables can be used to change the operating mode of the system implementation. For example, if you want to port an application from the operating system and wants THEOS w/32 programs operate in the same way, load the following variable:

B_EMULATE=THEOS
Variables Execution System, as B_EMULATE be loaded into a special initialization file. When the application starts, the system looks for a file execution with the same name as the application with the. W32 in the Windows directory (usually \ Windows). If the initialization file for a specific application does not exist or can not be found, will be used instead the default file W32APBLD.w32. (Any program chained or bound to always use the same main initialization program that is using the main program.)
The initialization file is editable W32APBLD.w32, is a text file that loads all environment variables to null or empty strings, so as to use the defaults. Edit the file (or preferably a copy) to load any variable to a specific value. Its content should not be replaced when installing a software update.
For a list of available environment variables, run OBW32 without any parameters. These appear grouped by function.
Variables related to the compiler and runtime system
These variables indicate the location of the compiler, runtime system and its components, thus avoiding having to update the autoexec.bat file.
B_CETBIN
Indicates the location w/32 compiler. By default this variable is updated: B_CETBIN=C:\CETBIN

B_CETINC
Indicates the location of the runtime components cet. Default is current directory cetlib \ INC: B_CETINC=C:\CETLIB\INC
B_CETLIB
This variable indicates the location of components cet win. Default is current directory c: \ cetlib. B_CETLIB=C:\CETLIB
B_BRTMDIR
This variable indicates the location of the system running WIN CET. Default is current directory c: \ windows \ system: B_BRTMDIR=C:\WINDOWS\SYSTEM

Variables de entorno relacionadas con ficheros y entradas/salidas
B_CSHFCLOSE
Load this variable to 1 to close all files before running a sentence CSH / CSI. This variable is subject to the default which is not close open files.
B_FPATH
Load this variable to the list of directories where different locations will pick the OPEN statement before launching a file not found error. B_FPATH supports a string of up to 255 characters. Separate each path with a semicolon symbol (;). For example, if the file "question" to be opened. BASIC first searches the current directory. If this file is not in this directory and loaded B_FPATH variable as below, then the program will try to open the file "D: \ FPA \ question".
B_FPATH= D:\APF
Note that if the variable is loaded B_FPATH search will be conducted for all open files except SEQUENTIAL OUTPUT mode. In this case, the runtime system will always look for the file in the current directory.
B_LIBFNTYP

Load this variable to 1 so that the file name in the format THEOS (abc) are open as \ b \ a \ c. If this variable is not set to 1 BASIC will try to open the file as. \ A \ b \ c.
B_LKPATH
Load this variable search paths used by executable files called by the judgments LINK, CHAIN, and RUN. . Separate each path with a semicolon symbol (;). Note that behaves like Windows w/32 to find an executable file. The search sequence is:
The directory where the calling program is executed.
The current working directory
The \ Windows or \ Windows \ system
The directories set in the PATH
The directories set within the variable B_LKPATH
B_REOPENS
Tells the OPEN statement the number of retries to access an indexed file. By default only try it once.
Variables affecting the use and control of printing
B_FFPTR
If set to one, it shall salo page when you close the printer (provided the printer channel is not open to the LOCK option - from version 9.32 b). The default is not to do the page break.
B_MAPPRTn
Select the name of the file containing the control sequences for a specific printer n, where n is the number of printer. The following example shows that the escape sequences that functions with PRINTER1 CRT is used in the file hplaser.lpr:
B_MAPPRT1=HPLASER.LPR
Note: Printing under Windows it is in implementation notes in the appendices of this manual. For more information on using B_MAPPRTn, see chapter formatted output in the Language Reference Manual CET BASIC.
B_PRINTERn
Check this variable with a specific printer as the Windows printing system in use (B_WINPRINT = 1). You specify a particular printer by:
B_PRINTERn = printer name
Under Windows NT and 95, this name should be assigned to the printer's "Print Manager". Under Windows, you have no option to give a name to a printer, so use the name of the driver installed on the port (for example, "HP LaserJet III").
Note that the only case where the printer number from 1 to 9 in the name of the variable is associated with the LPT port for that printer so that you can continue using the feature B_MAPPRTn. If the variable has the

value DEFAULT B_PRINTERn or DIALOG, there is no association between the number of printer and LPT port in any Windows environment.
The variable can also be enabled to make the screen a print dialog that allows the user to select a printer from the list specified in the "Print Manager".

B_PRINTERn=DIALOG

Since at present still be printing ASCII text, there will be some print dialog options as "copies" and the selection of specific numbers of pages that are disabled. (To be implemented when there is genuine Windows printing pictures in next releases).
It will use the system default printer if B_PRINTERn is equal to:
B_PRINTERx = DEFAULT
Print is described in detail in the Implementation Notes in the Appendix.

B_WINPRINT

Set the value to 1 if you want to use the new printer interface described in B_PRINTERn. B_WINPRINT If the variable is not set, then use the print method type. "Bat". Both methods are described in Notes Implementation.
B_WINPRINT = 1

Variables que afectan a la pantalla

B_DFLTBGC

Load the value of the default background color, use a program that when executed.
During the initialization process, the Framework will verify the value of this variable in W32APBLD.w32 file. If this variable is not loaded, l see a black background, before any color you use your program. If you use a background color in your program, we recommend loading this variable with the same color, this will prevent discoloration often undesirable. If you do not use color in your program, the gray will give a totally Windows. This can be done by setting the variables of foreground and background to black and white, ie the values 0 and 7 respectively.
To display a gray background:
B_DFLTBGC = 7
For more details on using color, see COLOR statement in Chapter Features Color and Line Drawing Manual CET BASIC Language Reference.

B_DFLTFGC

Load the value of the foreground color, for the same reason as discussed in the previous paragraph we recommend loading this variable with the same value you use in your application program. For example, the following variable will cause the foreground color is black.
B_DFLTFGC = 0
For more details on using color, see COLOR statement in Chapter Features Color and Line Drawing Manual CET BASIC Language Reference.

B_SCRNOLINES

Load the number of lines to be displayed on screen. The default is 25 unless B_EMULATE loaded in this case will be 24.

Variables que proporcionan compatibilidad con THEOS

The use of the following variables will allow their w/32 programs run as if they were running under the operating system THEOS.

B_<driver>DRIVE

Turn on to affect the way we interpret the physical drive THEOS driver. For example:
SET B_SDRIVE = C: \ TREND
The following program can be used to access c: \ trend \ dms \ data \ dms01:
$ lib = "DMS"
$ drive = "S"
open1: LIB $ & ". DATA.DMS01:" & DRIVE $, direct update
Activating these variables change the default behavior is: first find the file in the current directory, then the path specified in B_FPATH.

B_DOSTAB

Select to LINPUT USING statements that return a value of 9 for the INP TAB character as low THEOS ago. Otherwise, the default is to ignore the TAB.

B_EMULATE

Loading this variable to 1 mimics several features of THEOS, automatically activating the emulation-related variables such as B_FFPTR, B_GINVERT, B_OPTLOCK, B_THCONV, B_THLOCK, B_THPON and B_THVAL.

Enabling emulation attributes THEOS will screen control function as in THEOS THEOS file formats are recognized, that INP values are as expected, and that the escape character will become 27.
Note that division by zero errors will be detected. This is an illegal operation (in C) for good reason: if you were allowed the program may produce incorrect data without there being any notification.

B_OPTLOCK
When set to one, will cause the READNEXT, and function as in READPREV THEOS after a RESUME. (These sentences will run without moving the pointer to the file, registry retrying reading). If a record is locked, and this variable is not set, CET BASIC will read the next record, or earlier when running a RESUME.

B_THLOCK
Enabling this variable to 1 the program is suspended when it detects a stall condition. CET BASIC suspends execution only when faced with a judgment or ON ERROR ON LOCK.
If this variable is set, execution of an ON ERROR statement not generate an error when run blocking access statements that direct or indexed files. Refer to chapter "system error handling" the Language Reference Manual.

B_THPON
Activate this variable to one to reverse the default meaning PON / POFF as THEOS. CET BASIC normally shows low-intensity text under MS DOS and Windows environments. Jogging B_THPON ensure the variable use of high-intensity attribute in the foreground.

B_THVAL
Check this one variable to VAL functions and work as THEOS NBR. This will make judgments as having below zero:
PRINT VAL ("abc")
If not used the environment variable, this statement would generate a runtime error. See the Language Reference Manual CET BASIC to see the standard way of dealing CET VAL functions and NBR.

Generic Environment Variables

B_4DYEAR
Load this variable to 1, so that the work functions and DTE DATE with 4 digits.

B_DATEFORM
Charge 2 (European), or 3 (International) to change the default date format, showing dates in American format (MM / DD / YY). The CET BASIC DATEFORM option is used to change the date format in the program.

B_LANGUAGE
Introduces two 2-characters that specify the language, to force the function $ YESNO generate the appropriate text for the languages French (FR), German (GE), Swiss (SW), Finnish (FI), Spanish (SP) , Portuguese (PO) or Italian (IT). The default is English (EN).

B_ODBCERR
Put the value one, to help you get on screen error messages that may occur when accessing files through ODBC w/32.

B_YR2000
Check this variable with the last two digits of the year that will be interpreted as 20XX. This feature allows you to have dates of 1900 and 2000 without having to use a four-digit notation.
For example, suppose you have no data prior to January 1, 1925. In that case it could do:
B_YR2000 = 24
This would cause the judgments DAY ("01/01/24") and DAY ("01/01/2024" were identical.
Using this example, the following dates would be interpreted as:
01/01/00 01/01/2000
31/12/24 31/12/2024
01/01/25 01/01/1925
31/12/99 31/12/1999

B_CREQTAB
In the dialog box moves to the next control pressing ENTER.

B_OLDGETENV
Allows reading and writing of any environment variables either reserved or entered by the user.

```
Rem - The variable B_OLDGETENV = 1 must be active
Rem - in WINDOWS directory W32APBLD.w32
call cForceExit
Call Bgetenv (addrof ($ value), "B_USER")
Msgbox "Value B_USER:" & $ value
```

Rem Now change the value of b_user
Call Bputenv ("B_USER = 2")

B_MAXROWS y B_MAXCOLS
In programs that do not use graphical dialogs you can change the number of rows and columns, respectively, through the values assigned to these variables.
Default text screens CET W32 can show 24 rows and 80 columns of characters, if you change this setting to use a font kind which show the full number of rows and columns.

For example, if you set the following values in the environment variables file:
B_MAXCOLS = 133
B_MAXROWS = 26
You must use a font that can display 132 columns and 25 rows
B_DOTEQCOMMA (From 9.31 u version)
Allows you to press the point. on the numeric keypad is dispatched, the keyboard buffer.

1.7 Installation of drivers dongle (or keys Protection) w/32 for Windows NT, 2000, XP

There are two types of dongles as the destination of the same: USB, Parallel. The difference is in the installation.
For parallel dongles port:
If you are compiling and / or running programs under Windows NT, you must install the drivers of the Key Protective CET. (No special drivers are required under Windows 9x)
The drivers of the protection key, are included in the installation CD in the "\ Drivers \ Driver Pills (NT-2000)".
For USB dongles port:
You just have to install the drivers located on the CD in "\ Drivers \ Driver usb tablet" and then put in your port.

1.7 Key protection activation

For paralels dongles:
1. Download the driver from the direction http://www.omegasystems.com/utiles/ntdrvr.exe
2. Run the self-extracting downloaded to decompress the Sentinel.
3. From the folder where you unzipped the Sentinel, run SETUPX86.EXE, then enter the menu functions, to press "Install Sentinel Driver".
4. For the changes to take effect you must restart the machine.
For USB dongles:
After installation of the drivers, the activation is automatic.

1.8 MACADDRESS CET Protection

The alternative to the backpack (or dongle) is a code related to the MAC address of the network card of the PC. To activate this system cetfmac.exe execute the command, which returns the address of the card, as follows:

```
C:\>cetfmac
CET MAC Address Finder
----------------------
Found 1 MAC Addresses:
MAC Address  0 : 00051C0DAA93

C:\>
```

This value corresponds to a code supplied by the staff of our office.
These values along with other staff supplied by Omega conform system authentication. These values must be set in CET environment variables within W32APBLD.w32 file. To do this you need to edit the file and find the following block of variables in it, and set the corresponding values. The characters '*' mean comments online, so I will delete those involving one of these variables. The purpose of these reviews is not to lose the variables when using a backpack system, and in turn not interfere with.

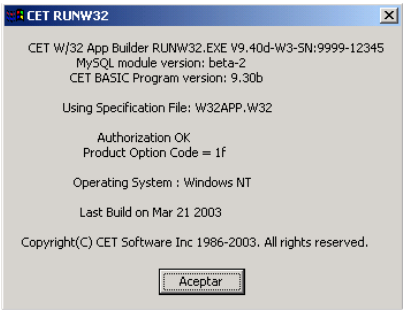
* REM network address (MAC address) Mac RED
* B_MACAD = 00051C0DAA93
* Fixed REM
* B_MACPC = 133

* REM serial n §
* B_MACSN = 999
* REM n § of concurrent users in MSDOS or Windows
* B_MACNU = 10
* REM options 1 = POSWINDOWS
* B_MACLO = 7
* REM n § Access user
* B_MACNA = 0
* REM compiler password for win
* B_MACPW = NFLZTL

1.9 Verifying the Installation under Windows

To determine if the package w/32 CET has been installed correctly, insert the key (dongle) CET at the parallel output, open the MS DOS or enter the following:
Runw32 or obw32 V
The command should show the serial number in use.

Valid


CET RUNW32
CET W/32 App Builder RUNW32.EXE V9.40d-W3-SN:9999-12345
MySQL module version: beta-2
CET BASIC Program version: 9.30b
Using Specification File: W32APP.W32
Authorization OK
Product Option Code = 1f
Operating System : Windows NT
Last Build on Mar 21 2003
Copyright(C) CET Software Inc 1986-2003. All rights reserved.
Aceptar

Not valid

CET RUNW32
CET W/32 App Builder RUNW32.EXE V9.40d-W3-SN:0-0
MySQL module version: beta-2
CET BASIC Program version: 9.30b
Using Specification File: W32APP.W32
Authorization FAILED
Operating System : Windows NT
Last Build on Mar 21 2003
Copyright(C) CET Software Inc 1986-2003. All rights reserved.
Aceptar

1.10 Web site

We have a website in www.omegasystems.com, which has information about us and our products.
On the left are the links to the different sections of our website:
About us: refers to our location, functioning, objetovos, and staff.
CET Basic: Explains the adjectives of language, history, and usefulness. From this section you can go to the download area of CET with the link included.
Applications Available: Index of programs created with their functions.
4GL Generator: Here is a summary of our work herramiente.
Uniphone: Indicates the options and features Uniphone system.
There is also a helpful area where you can download the exhibits.

1.11 Versions

CET, throughout his life, has evolved, resulting in a number of versions, the most recent are: 9.31 (m, K, z), 9.32 (b, m, u), 9.40 (c, d) , 9.41 (a).
9.50, 9.52, 9.65 and 9.72

Chapter 2 Overview

2.1 Introduction

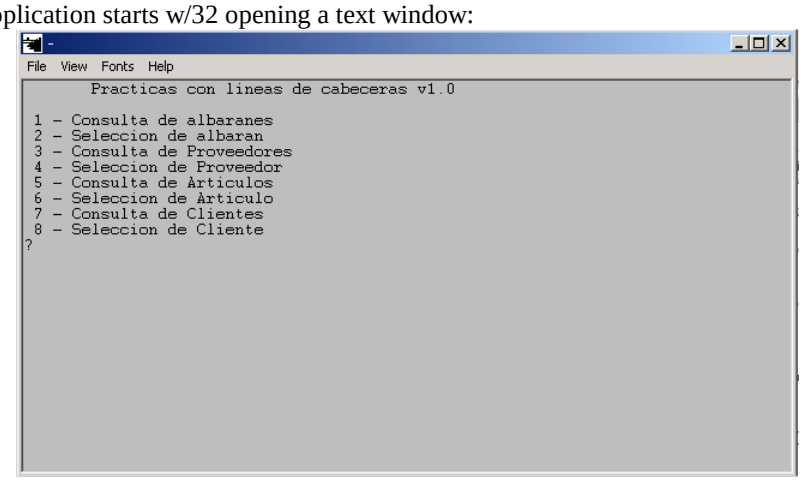
The Application Development System w/32 CET is designed for commercial applications for Microsoft Windows operating systems. W/32 differs from most development packages for MS Windows that provides the tools to move existing programs methodically 3rd generation to modern applications of graphical user interface (GUI).

2.2 The base language

Microsoft and other companies market development tools have rediscovered the BASIC as the best language to use routines, macros and programs quickly. BASIC language has grown in sophistication and power, while maintaining ease of use.
The CET Application Development System w/32 also used as the base language BASIC. CET BASIC is a language rich with excellent facilities with which to develop commercial applications.
CET BASIC is easy to learn, and expand. Case still running very complicated operations have an easy syntax.
CET BASIC includes sophisticated control structures, a large number of data types, all forms and types usual file operations, hundreds of string handling functions, data conversion, screen handling, formatted output, capture and recovery errors.
Unlike many interpreted languages, CET BASIC provides high performance at runtime. The compiler generates code w/32 32-bit highly optimized, allowing applications to run natively under Windows 9x, Windows NT, 2000 and XP.
CET BASIC is management oriented and includes features such as:
Sophisticated operations with sequential files, direct and indexed, working with level locks files and records, detecting and controlling locks shared files. What makes programming in multiuser environments.
Supports data types decimal and floating point arithmetic.
Full support matrix arithmetic and E / S files.
Types type COBOL formats to display fields on screen and print listings.
Chaining programs to generate structured programs and applications easier to maintain.
Comprehensive error detection and capture.
Easy access to operating system commands and liaison with external routines in C, BASIC and assembler.
Full support preprocessor C.
Supports Multi-platafomas under Windows (all versions) and Linux.
CET BASIC has support multinational. Provides complete control over the numeric and date formats. Error messages are stored externally, which facilitates translation.

2.3 The W/32 environment.

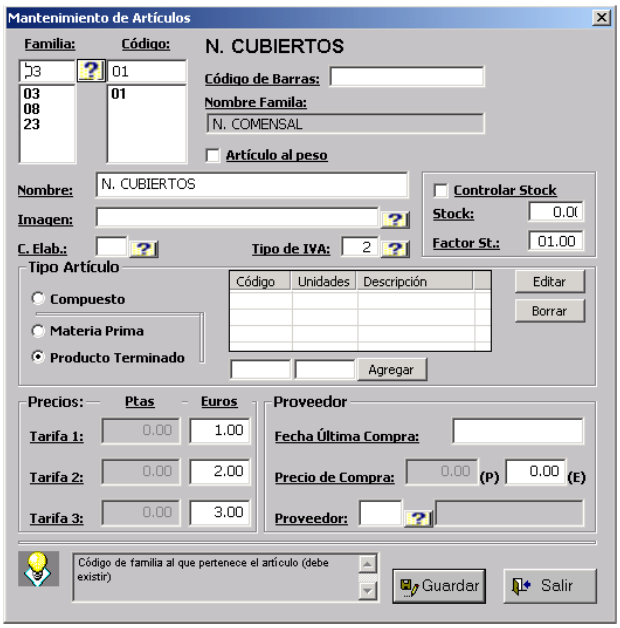
The compiler takes w/32 CET BASIC conventional applications and links (linka) within the Windows environment for 32-bit. This means that existing programs in CET BASIC can be compiled and executed immediately without modifications, and then gradually improve specific characteristics including Windows.
By default, an application starts w/32 opening a text window:


File View Fonts Help
Practicas con lineas de cabeceras v1.0
1 - Consulta de albaranes
2 - Seleccion de albaran
3 - Consulta de Proveedores
4 - Seleccion de Proveedor
5 - Consulta de Articulos
6 - Seleccion de Articulo
7 - Consulta de Clientes
8 - Seleccion de Cliente
?

The text window provides a familiar screen character style, ready for them to run existing programs CET. The menu command, by default, has the following options: selecting typefaces (fonts, or fonts), appears or not the toolbar that appears or not the status bar and a help window.
Development System w/32 CET provides powerful tools to enhance your applications without having to modify the source code. By default, all applications are linked to the framework by which you can control the fonts (fonts), the number and types of elements, displaying the status bar and toolbar. The `Manager 'can be modified (eg. Selecting a specific source) these new features will be used automatically whenever you compile a program. To the framework, it devotes an entire chapter in this manual.
You can use a special compiler flag that can be used to change their existing input fields created LINPUT, USING LINPUT controls and GET for Windows text editors to quickly acquire your application windows look.
See Chapter w/32 Compiler for more information on the use of these special text editors. The Windows Framework described in this chapter.

2.4 The W/32 IDE.

The Application Builder w/32 has a graphical development environment, (W32APBLD.exe) that has multiple uses for convenient development of their applications. While this program you can create:
Projects to build, test and maintain your application executable.
Source files using a text editor especially designed for writing for long programs. The debugger is also available to expedite the debugging phase.
User Interfaces (GUI) that consist of a collection of windows that will give you a much more visual for your application.
Due to the variety of features it offers, has been explained in three separate chapters: The Integrated Development Environment w/32, w/32 Debugger and Editor w/32 Dialogues.
The following screen is an example display format, illustrating some of the features that will be explained in the chapter w/32 Dialogues Editor.



Both the display format as the dialog box consists of a group of controls to draw on screen or text, or enter information from the user. Each type of control has a set of properties defined behavior.
A display format is a GUI input that is used to replace the text-based screen of the application. Although the window may be full screen or any size that is defined with the dialog editor, can be moved or resized by the user.
While the display format is active, you have full access to the menu bar and / or toolbar buttons. A format can be minimized or maximized.
A dialog box is similar to a display format except that the dialog should be closed (eg "clicking" OK or Cancel), before they can work with the rest of the application.
You can create a variety of dialogues. A dialog box is a simple message box that pops up a text and waits for the operator to select a button. A complex dialog box will release a large number of fields per screen, waiting for the operator to enter via the mouse or keyboard.
Basically, the dialog boxes may be used anytime you want to get information to or from the operator without affecting what is currently on screen, or other active dialog boxes. On other operating systems, may have

opened a window (such as Phase One), drawn text, introduced by LINPUT information, and finally closed the window to continue. Under Windows, this will typically with a dialog box.
Whether you are working with a display format, or a dialog box, the dialog editor generates a BASIC program that can be modified later.

2.5 Subroutines BASIC

Development System w/32 CET provides special statements that can be used to link (linkar) program modules that call BASIC subroutines within a single executable. For example, the head of a subroutine has the following format:
$ SUB module-name
$ SUB defines a subroutine that can be called from the `main 'from other subroutines BASIC or BASIC.
There are two rules for sharing BASIC variables:
Any variables referenced within the same module or program source (file. B) refers to the same variable. This is evident.
Any variable that is used in two modules of different sources are different, unless defined by $ global judgment. This will be true even if two variables have the same name.
The variables that either are completely independent module (unless specifically defined as global), allows you to write and compile separate BASIC subroutines, and then link them (linkarlas) in a single executable, without having to worry about possible collisions between variable names separate modules.
Thus, new sentences are introduced as $ GLOBAL BASIC to define global data, $ CLEAR to clear all variables and dimensioned string tables, and $ EXIT is used to return a BASIC subroutine.
The ability to use BASIC subroutines is another powerful feature of Application Development System w/32 CET to which he devotes a separate chapter in this manual.

2.6 Mouse support

The mouse input device is the most important data for Windows applications. It gives the user an intuitive way of selection and implementation.
Although entry by mouse is usually difficult to integrate keyboard oriented applications, w/32 provides a powerful way to use the mouse at any time.
The mouse handler detects any movement or touch and responds immediately by calling specific BASIC subroutines such as:

- cetLBtnDn
- cetLBtnUp
- cetLBtnDClk
- cetRBtnDn
- cetRBtnUp
- cetRBtnDClk
- cetMouseMove

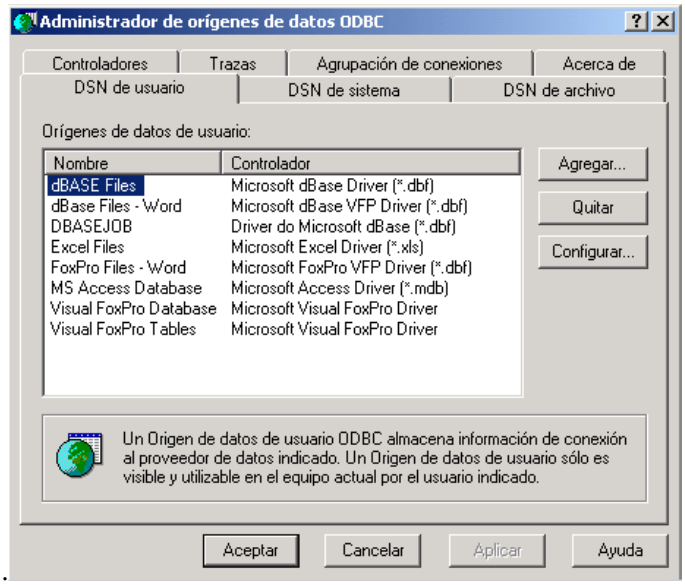
These subroutines are invoked when the mouse buttons are pressed, released, or doubly pressed and when movement is detected in the mouse. Your program can perform any action when it detects these mouse events or define global variables that will later be used in other routines.
Even existing applications that use INPUT or GET LINPUT for user input can be enhanced by using the mouse. Using a feature called cInputQ Wi/32, a program can preload the keyboard input queue with answers. Using this method, a mouse event can force entering characters queued input keyboard, making LINPUT or INPUT statements "give input" to these characters later.

2.7 Viewing graphic images.

W/32 provides routines to display, position and resize graphics (in. Bmp,. Dib or. Jpg using cetJPEG function) while running an application w/32. This feature allows you to view photos, logos and other graphics in a window within the same program.

2.8 ODBC support

The easy connection to open databases (in English ODBC - Open Database Connectivity -) is a standard created by Microsoft that provides transparent access from your applications to a large number of database systems and file managers.
ODBC drivers available for MS Windows include file system administrators (eg, dBase, Btrieve) Database administrators (eg, FoxPro, Paradox and MS Access) and SQL servers like SQL Watcom SQL and Gupta. It is possible to give full support to all relational databases through ODBC drivers, existing MS SQL Server, Oracle, Informix, Sybase and others.
To use ODBC in a program, you must have DOBC driver installed correctly on your system and data source configured according to the documentation for the driver. So w/32 program allows the user to specify a data source or open source directly via the OPEN statement



You can then use the READ statement to select the set of rows (records) you want from the table open (file). The names of the columns (fields) of the table will be loaded as fields in a record, which can be used to further interrogate the database using READs. The judgment READNEXT used to access each row that has been read. MATWRITE WRITE statements and can be used to update the data in the table.

2.9 MySQL

2.9.1 Introduction

This document describes the optional module for MySQL runtimes CET w/32 of BASIC.

2.9.2 New features

The optional module allows the programmer MySQL CET BASIC access MySQL tables like ISAM files.

The module exports a set of APIs functions similar to the MySQL C APIs. These functions allow programmers to use the full power of the MySQL database in BASIC programs.

2.9.3 Non supported features

The following w/32 compiler are not yet supported with MySQL:

- The functions ACCESS Phase One (FREEDOM).
- Profits blist, bcreate, bcopy and similar.

2.9.4 References

- Paul DuBois. MySQL, Segunda Edición. (Pearson Education, Enero 2003. ISBN 0735712123)
- Michael Widenius, David Axmark. MySQL Reference manual (O'Reilly & Associates, Junio 2002. ISBN 0596002653)
- Documentos .DOC que se pueden descargar de la web MySQL http://www.mysql.com/documentation/index.html

2.9.5 Working with MySQL data base.

The MySQL module provides BASIC programmers CET two ways to work with the MySQL database:

- Emulation indexed files
- Native APIs.

2.9.6 Files indexed emulation.

The extension module allows access to tables of MySQL database, using standard statements of CET BASIC file accesses:

- OPEN
- CLOSE
- READ
- READNEXT
- READPREV
- WRITE
- DELETE
- UNLOCK
- MAT READ
- MAT READNEXT
- MAT READPREV
- MAT WRITE

And end of file:

- EOF

2.9.7 Configuración

The form of emulation is set by the variable contained in w32appw32 B_MYSQL:

B_MYSQL = <server>: <user>: <palabra of Step>: <database>

where:

<server> server name or IP address

<user> the user name to connect to the MySQL server

<palabra of Step> password that corresponds to the user name above

<database> database used for the emulation module

The emulation module works only with a single database. This database must contain a table called _cet_files_ configuration. This table contains information about the files indexed BASIC converted to MySQL tables. This information includes the name of the original file, the MySQL table name, key lengths and registration, etc.. The table format is described below under the heading "Installing and configuring the MySQL Server".

2.9.8 File access methods and formats

The module supports only binary indexed files. Valid options for the OPEN statement are:

- INPUT INDEXED
- OUTPUT INDEXED
- UPDATE INDEXED
- INPUT INDEXED LOCK
- OUTPUT INDEXED LOCK
- UPDATE INDEXED LOCK

2.9.9 File names

The configuration table contains information _cet_files_ mapping file names indexed table names. The emulator uses the alphanumeric file name is used in the sentence OPENcomo a search key for the table _cet_files_ and can locate the table name and other information required for emulation / out.

The emulator does not support the location of files in different directories, in fact uses the filename literally. This means that if you have two or more files of the same name indexed located in different directories when transferring these files to the MySQL database, you must choose different names for tables representing these files. The mapping will look like this:

Directory1 \ Data -> Data1

Directory2 \ Data -> Data2

That is, the OPEN # 1: "Directory1 \ Data" table opens Data1 and

the OPEN # 1: "Directory2 \ Data" Data2 table opens.

The emulator also use the configuration table to determine if any particular OPEN statements open a table in the database, or what opens is a native ISAM file. This allows the programmer to move BASIC ISAM file data to the MySQL database file by file.

2.9.10 Files and block records

The emulation module supports file locking (LOCK option in the OPEN statement) and record locking when the file is opened in update mode.

2.9.11 Natives APIs

The emulation module exports a set of functions usable from CET BASIC with similar functionality to the C APIs of the MySQL client library. These APIs can be used to access the MySQL database in the most efficient language itself using native MySQL server - SQL.

2.9.11.1 Native APIs functions references

The table that follows C benchmark MySQL functions C, and its equivalent in BASIC. Refer to the MySQL API documentation for details.

C Function MySQL	cetmysql Function
mysql_affected_rows (Returns the number of rows changed / deleted / inserted by the last UPDATE, DELETE, or INSERT)	`call cetmysql.affected_rows ( mysql%, addrof ( rows% ) )`
mysql_close (Close connection on server)	`call cetmysql.close ( mysql% )`
mysql_connect	`No implementado, use real_connect`
mysql_change_user (Changes user and data base associated to an open conecction)	`call cetmysql.change_user ( mysql%, user$, passwd$, db$ | db%, addrof ( errno% ) )`
mysql_character_set_name (Returns the name of the default character set for the connection)	`call cetmysql.character_set_name ( mysql% addrof ( name$ ) )`
mysql_create_db	`Not implemented, use CREATE DATABASE`
mysql_data_seek (Find an arbitrary row in the result set of a query)	`call cetmysql.data_seek ( mysqlres%, offset% )`
mysql_debug	`Not implemented`
mysql_drop_db	`Not implemented, use DROP DATABASE`

mysql_dump_debug_info	Not implemented
mysql_eof	Not implemented, fetch_row_and_lengths mysqlrow% returns 0 when the condition is detected eof
mysql_errno	Implicitly called by all functions that can set the value of errno and the result is returned in the argument of the function errno%.
mysql_error (Returns the error message of the most recently invoked MySQL function)	call cetmysql.error (mysql% addrof (error$))
mysql_escape_string	Not implemented
mysql_fetch_field	Not implemented
mysql_fetch_field_direct (Returns the type of a field in a table given its number)	call cetmysql.fetch_field_direct (mysqlres%, fieldnum%, addrof (name$), addrof (table$), addrof (def$), addrof (type%), addrof (length%), addrof (max_length%), addrof (flags%), addrof (decimals%))
mysql_fetch_fields	Not implemented
mysql_fetch_lengths	Implicitly by the function call fetch_row_and_lengths.
mysql_fetch_row_and_lengths (Go to the next row in the result set)	call cetmysql.fetch_row_and_lengths (mysqlres%, addrof (mysqlrow%), addrof (lengths%), addrof (errno%)) call cetmysql.fetch_field_value (mysqlrow%, lengths%, field%, addrof (value$)) Use fetch_field_value para acceder al valor del campo de la fila actual.
mysql_field_seek	Not implemented
mysql_field_count	Not implemented
mysql_field_tell	Not implemented
mysql_free_result (Frees memory used by a result set)	call cetmysql.free_result (mysqlres%)
mysql_get_client_info (Returns information about the client version)	call cetmysql.get_client_info (addrof (info$))
mysql_get_host_info (Devuelve una cadena describiendo la conexión)	call cetmysql.get_host_info (mysql% addrof (info$))
mysql_get_proto_info (Returns the protocol version used by the connection)	call cetmysql.get_proto_info (mysql% addrof (info%))
mysql_get_server_info (Returns the server version number)	call cetmysql.get_server_info (mysql% addrof (info$))
mysql_info (Devuelve información de la consulta ejecutada más recientemente)	call cetmysql.info (mysql% addrof (info$)

	)
mysql_init (Initialize MySQL estrcutura)	call cetmysql.init (addrof (mysql%))
mysql_insert_id	Not implemented
mysql_kill	Not implemented
mysql_list_dbs	Not implemented, use SHOW DATABASES
mysql_list_fields	Not implemented, use SHOW COLUMNS
mysql_list_processes	Not implemented
mysql_list_tables	Not implemented, use SHOW TABLES
mysql_num_fields (Devuelve el número de columnas de un conjunto de resultados)	call cetmysql.num_fields (mysqlres%, addrof (fields%))
mysql_num_rows (Devuelve el número de filas de un conjunto de resultados)	call cetmysql.num_rows (mysqlres%, addrof (rows%))
mysql_options	Connection Time in seconds call cetmysql.opt_connect_timeout (mysql%, timeout%, addrof (errno%)) Use compressed protocol client / server call cetmysql.opt_compress (mysql%, addrof (errno%)) call cetmysql.opt_named_pipe (mysql%, addrof (errno%)) Command to execute when connecting to MySQL server call cetmysql.init_command (mysql%, command$, addrof (errno%)) Read options specified file instead of my.cnf call cetmysql.read_default_file (mysql%, file$, addrof (errno%)) call cetmysql.read_default_group (mysql%, group$, addrof (errno%))
mysql_ping (Check if the connection to the server is active)	call cetmysql.ping (mysql%, addrof (errno%))
mysql_query	No implementada, use real_query
mysql_real_connect (Try to establish a connection to a database engine MySQL is running on the specified host)	call cetmysql.real_connect (mysql%, host$ | host%, user$ | user%,

	passwd$ | passwd%, db$ | db%, port%, unix_socket$ | unix_socket%, flags% addrof (errno%))
mysql_real_query (MySQL executes the statement specified)	call cetmysql.real_query (mysql%, query$, addrof (errno%))
mysql_reload	Not implemented
mysql_row_seek	Not implemented
mysql_row_tell	Not implemented
mysql_select_db (Select the specified database such as the database current connection)	call cetmysql.select_db (mysql%, db$, addrof (errno%))
mysql_shutdown	No implementada
mysql_stat (Returns a string containing statistical information related to the activity on the server)	call cetmysql.stat (mysql%, addrof (stat$), addrof (errno%))
mysql_store_result (You must call this function whenever a query to retrieve data)	call cetmysql.store_result (mysql%, addrof (mysqlres%), addrof (errno%))
mysql_thread_id	Not implemented
mysql_thread_safe	Not implemented
Mysql_use_result	call cetmysql.use_result (mysql%, addrof (mysqlres%), addrof (errno%))

2.9.12 Example of Using native APIs CET BASIC programs.

In the distribution includes the mysqlapi.b program.

Compile the program:

mysqlapi.b bl

and run:

mysqlapi _cet_files_

If the variable is set correctly B_MYSQL, the program connects to the MySQL server and draw on screen the contents of the table _cet_files_. Spend any table name to the program. If the table name does not exist, or if the program encounters any errors, the screen will bring a message and exit the system.

The program uses existing definitions in the variable B_MYSQL for the MySQL host name, user name, the password of the user and the database. The program expects the name of the table to be treated, to be passed through the command line.

```
10 def fnerr% ( msg$, mysql%, errno% )
20
30 fnerr% = errno%
40
50 if errno% <> 0
60  call cetmysql.error ( mysql%, addrof (error$) )
70  print msg$; " error"; errno%; "["; error$; "]"
80 ifend
90
100 fnend
110
120 table$ = cmdarg$ (1)
```

```
130 if len ( cmdarg$ (1) ) = 0
140 msgbox "Uso: mysqlapi <nombre tabla>"
150 quit
160 ifend
170
180 call Bgetw32 ( ADDROF (b_mysql$), "B_MYSQL" )
190 print "B_MYSQL='" & b_mysql$ & "'"
200 if LEN (b_mysql$) = 0
210 msgbox "B_MYSQL no está definido”
220 quit
230 ifend
240
250 host$ = ""
260 user$ = ""
270 password$ = ""
280 db$ = ""
290
300 start%=1
310 pos% = sch ( start%, b_mysql$, ":" )
320 if pos% <> 0
330 host$ = b_mysql$[1:pos%-1]
340 start% = pos%+1
350 pos% = sch ( start%, b_mysql$, ":" )
360 if pos% <> 0
370 user$ = b_mysql$[start%:pos%-1]
380 start% = pos%+1
390 pos% = sch ( start%, b_mysql$, ":" )
400 if pos% <> 0
410 password$ = b_mysql$[start%:pos%-1]
420 db$ = right$ ( b_mysql$, pos%+1 )
430 ifend
440 ifend
450 ifend
460
470 print "Host='" & host$ & "'";
480 print "  Usuario='" & user$ & "'";
490 print "  Palabra de Paso='" & password$ & "'";
500 print "  Base Datos='" & db$ & "'";
510 print "  Tabla='" & table$ & "'"
520
530 query$ = "select * from " & table$ & " limit 5"
540
550
560 call cetmysql.get_client_info ( addrof ( info$ ) )
570 print "Cliente "; info$
580
590 call cetmysql.init ( addrof (mysql%) )
600 print "mysql"; mysql%
610 if mysql% = 0 then quit
620
630 call cetmysql.real_connect ( mysql%, host$, user$, password$, db$, 0, 0, 0,
addrof ( errno% ) )
640 if fnerr% ( "real_connect", mysql%, errno% ) <> 0 then goto close
650
660 call cetmysql.get_host_info ( mysql%, addrof ( info$ ) )
670 print "Host "; info$
680
690 call cetmysql.get_server_info ( mysql%, addrof ( info$ ) )
700 print "Servidor "; info$
710
720 call cetmysql.get_proto_info ( mysql%, addrof ( info% ) )
730 print "Protocolo "; info%
740
750 call cetmysql.character_set_name ( mysql%, addrof ( name$ ) )
760 print "conjunto de carácteres "; name$
770
780 call cetmysql.real_query ( mysql%, query$, addrof ( errno% ) )
790 if fnerr% ( "real_query", mysql%, errno% ) <> 0 then goto close
800
810 call cetmysql.info ( mysql%, addrof ( info$ ) )
820 print "Info "; info$
830
840 call cetmysql.store_result ( mysql%, addrof ( mysqlres% ), addrof ( errno% )
)
```

```
850 if fnerr% ( "store_result", mysql%, errno% ) <> 0 then goto close
860
870 call cetmysql.num_rows ( mysqlres%,  addrof ( rows% ) )
880 call cetmysql.num_fields ( mysqlres%,  addrof ( fields% ) )
890 print "Resultado"; mysqlres%, "filas"; rows%, "Campos"; fields%
900 print
910
920 dim FieldName$ ( fields% )
930
940 print "Campo        Tabla.Campo       Def         Tipo  Lon  MaxL  Dc
Flags"
950 for fieldnum%=0 to fields%-1
960 call cetmysql.fetch_field_direct (  mysqlres%, fieldnum%, addrof ( name$ ),
addrof ( table$ ), addrof ( def$ ), ~
       addrof ( type% ), addrof ( length% ), addrof ( max_length% ), addrof (
flags% ), addrof ( decimals%) )
970 print using " ###  'RRRRRRRRRR.'LLLLLLLLLL 'LLLLLLL  ### ##### ##### ##
##########", fieldnum%, table$, name$, def$, type%, length%, max_length%,
decimal%, flags%
980 FieldName$ ( fieldnum%+1 ) = name$
990 next fieldnum%
1000
1010 wait
1020
1030 fetch:
1040 call cetmysql.fetch_row_and_lengths ( mysqlres%,  addrof ( mysqlrow% ),
addrof ( lengths% ), addrof ( errno% ) )
1050 if fnerr% ( "fetch_row_and_lengths", mysql%, errno% ) <> 0 then goto free
1060
1070 if mysqlrow% = 0 then goto free
1080
1090 print
1100
1110 for field% = 0 to fields%-1
1120 call cetmysql.fetch_field_value ( mysqlrow%, lengths%, field%, addrof (
value$ ) )
1130 if fnerr% ( "fetch_row_value", mysql%, errno% ) <> 0 then goto free
1140 print FieldName$ ( field%+1 ); " ["; value$; "]"
1150 next field%
1160
1170 wait
1180
1190 goto fetch
1200
1210 free:
1220
1230 call cetmysql.free_result ( mysqlres% )
1240
       call cetmysql.stat ( mysql%, addrof (stat$), addrof (errno%) )
     if fnerr% ( "stat", mysql%, errno% ) <> 0 then goto close
       print
     print "Estadisticas:"
       print stat$

1250 close:

1260
1270 call cetmysql.close (mysql%)
1280 quit
```

2.9.13 Installing and configuring the MySQL server

2.9.13.1 Installing the MySQL Server.

MySQL binaries for different operating systems are available on the web page of MySQL www.mysql.com. We recommend downloading the version 4.0, which is also recommended by MySQL for new development itself. During the development of this module was used version 4.0.10-gamma. There is a need for emulation InnoDB ISAM files, so select the server configuration that includes InnoDB.

Please follow the instructions on the Web page to download and install the server. The server can be installed in any directory. The next chapter "Configuring the MySQL Server" assumes that it is installed on a Windows machine in the directory C: \ MYSQL.

Probably should add C: \ mysql \ bin to your PATH so you can easily run the command-line tools that brings MySQL.

It is also recommended, download and install the "MySQL Control Center".

2.9.13.2 Configurando el servidor MySQL

2.9.13.3 Configuración específica del Servidor

The MySQL server uses the configuration file my.cnf. On Windows machines this file is usually located in C: \ my.cnf.

- The MySQL installer puts four examples configuration files within the root installation directory, C: \ MySQL in our case. Files are my-small.cnf, medium.cnf my-my-my-huge large.cnf and cnf. Select the appropriate file for your machine configuration and the size of the database. Refer to the MySQL documentation for more details.
- Copy the configuration selected configuration file in C: \ my.cnf
- Open the file c: \ my.cnf with any text editor and uncomment all configurations that follow the line that says "remove the comments that follow if you are using tables Innobases". ("Uncomment the Following if you are using Innobase tables".)
- Pay attention to the directories that are with innodb_data_home_dir variables, innodb_log_group_home_dir and innodb_log_arch_dir. These variables specify the location of the InnoDB data files. You can override the defaults with anything you want, or you can leave them as is.
- Create the directories above. The server creates these directories, should User Control yourself.
- Open the command line window, and start the MySQL server for the first time:

  mysqld-max-console

  The server creates the space for InnoDB data and draw some message while doing this:

  InnoDB: The first specified datafile c: \ ibdata \ ibdata1 did not exist:

  InnoDB: A new database has been created!

  .....

  011024 10:58:25 InnoDB: Started

  - Open the second command line window, and use the utility mysqladmin to shut down the server:

  mysqladmin-user = root shutdown

  - Install the server as a Windows service and start it:

  mysqld-max-install

  net start mysql

2.9.14 Configuration File Access emulator

- You should then set up users, and database. Although everything can be done with the mysqladmin utility, is easier to use the "Control Center" to do so. Please refer to the MySQL documentation and MYCC for details.
- You should create:

- - A new database for use by the emulator. Call it what you want. This name will be referenced in the variable B_MYSQL.
- - A new user with guaranteed access to the emulator database. The user name and the password must be referenced in the variable B_MYSQL.
- - A new table _cet_files_ in the newly created database. The table may be of any type. The type MyISAM is recommending. This table must contain the following columns:
- or Name char (255)
- or Type char (1)
- unsigned int or reclen
- unsigned int or keylen
- Table or char (100)
- or Key char (100)
- Record or char (255)
- - Define both the name and the type of the field as primary index

2.9.14.1 Example configuration script..

The sample script w32init.sql MySQL configuration supplied with the module. The script creates a database called w32 and an empty table _cet_files_ table within this database. The script does not create a new user, it is not necessary for the example configuration. You can use the root user to check the file access emulation. However, for the actual work, we recommend that you create a specific user for CET BASIC applications and allow this user to access only emulator database. The main reason for this is that the user name and the password file on your w32app.w32. The easiest way to create a user is running the utility MYCC.

To run the script, start the utility mysql as root:

```
mysql –-host=<host> --user=root –-password=<password>
```

and run w32init.sql sript

```
mysql> source w32init.sql;
Query OK, 0 rows affected  (0.01 sec)

Query OK, 1 row affected  (0.00 sec)

Database changed
Query OK, 0 rows affected  (0.00 sec)

Query OK, 0 rows affected  (0.00 sec)
```

Note that the script will destroy the database 'w32' if any. Please be careful when you run the script if you already have some files imported from the database..

2.9.15 Moving ISAM data files to MySQL tables

The method of transferring data from a file into a table isam MySQL, includes the following steps:

- Analyze the ISAM file with the utility bsqlconv
- Dividing the ISAM file or in two or more files if necessary.
- or Create tables in the database emulator and records within the table _cet_files_
- Export data file or a file isam utility running extended texts blist
- or Import data from a text file to a MySQL table

2.9.15.1 Tools

2.9.15.1.1 bsqlconv utility

The utility has been specially written bsqlconv to assist in the transfer of data isam-> mysql

The command line is:

bsqlconv [-f from_key] [-t to_key] [-c counter] <file_name>

file_name Specifies the name of the file in a DOS or THEOS format.

For example, either of the following names may be used to

list cp\data\custname:

.\cp\data\custname (or cp\data\custname)

cp.data.custname

-f from_key, Scans from check "desdeclave" to "hastaclave" key indexed file or the registration number in a direct file. By default starts from the beginning.

-t to_key, reaching analyzes the file "hastaclave" without including it. By default reaches the end of file.

-c counter, Specifies the number of records to be analyzed.

-v shows oftware version

The utility reads a file records indexed within a predetermined range of keys and analyzes the structure of each record. The main reason the utility is to detect inconsistencies in the format of the records. Only files with the same format in different registers, can be transparently converted to MySQL tables. If the file contains records with different formats, the file should be split into two or more files with the same format records. Thus such a file would only be moved in two or more tables MySQL, provided the BASIC program was also modified CET parallel.

If the utility does not find inconsistencies, then create and lift the screen a sequence of SQL statements that can be used to create the table MySQL table, add the record to the table _cet_files_ and import the data to the table.

If the utility find any inconsistency, will draw with a warning screen. The developer will then have to use the command line parameters-f,-t and-c to select only part of the original file for analysis.

2.9.15.1.2 Extended Utility blist

The utility blist is extended with the option-e within the command line. When used, this option will force the blist to generate output can be imported into the MySQL database. Ignore all page headers and page, enclose the field values in double quotation marks and separated by commas.

The-f,-t and-c can be used to list only parts of the file. If necessary may be detected it was necessary to divide the file. In this case these options generally should be used with the same values that were used with the utility bsqlconv.

2.9.16 Example

Here we see an example of the procedure for the conversion of ISAM indexed files with different record formats. This procedure assumes that the files will be moved to tables in a database called 'w32'que must already be created .. _cet_files_ Table must also be created in the way explained above..

2.9.17 Creating ISAM file to conversion

In the example we use an indexed file with the first two records in a different format than the rest of the file. First create the file with Bcreate:

bcreate example keylen indexed reclen 100 20

Then write data to the file using the following program in BASIC CET:

```
open #1:"example", update indexed
```

```
rem escribe los registros iniciales, cada registro contiene un
     campo numérico y otro entero.
write #1,"0001" : 1234.5678, 1234
write #1,"0002" : 8765.4321, 8765
rem escribe el resto del fichero con registros que contienen un
campo alfanumérico, otro entero y el último numérico.
for i%=1 to 9
    write #1,"Record" & STR$ (i%) : "Record number " & STR$
          (i%), i% * 100 + i%, i% * 1111.1111
next i%
close #1
```

List the contents of the file using the blist command:

```
blist example
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:::::: example  (Indexed)
Key    Length = 20
Record Length = 200
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:::::: 0001                :1234.5678,1234
0002               :8765.4321,8765
Record1            :Record number 1,101,1111.1111
Record2            :Record number 2,202,2222.2222
Record3            :Record number 3,303,3333.3333
Record4            :Record number 4,404,4444.4444
Record5            :Record number 5,505,5555.5555
Record6            :Record number 6,606,6666.6666
Record7            :Record number 7,707,7777.7777
Record8            :Record number 8,808,8888.8888
Record9            :Record number 9,909,9999.9999
```

2.9.18 Analyzing the isam file

Run the utility bsqlconv to discover the structure of the file:

```
bsqlconv example
```

Should shows following:

```
Input file: example
# Indexed file 'example' Key Length 20 Record Length 200
Record 'Record1' :Inconsistent type 4 of field 1. Expected 2
```

The message list by the utility means that the registration key file with "Record1" has a different format than all previous records in the file.

If the utility gets a message like this then the file will not become a single MySQL table. Should be used two or more tables. The first table will receive the records from the beginning of the file until the record preceding the record with key 'Record1'. Consider just this part of the input file:

```
bsqlconv example -t Record1
```

The utility will draw onscreen:

```
Input file: example
  To: Record1
# Indexed file 'example' Key Length 20 Record Length 200
No inconsistencies found
drop table if exists example;
create table example
 (
        Key CHAR (20) BINARY NOT NULL PRIMARY KEY,
        Fld1 FLOAT,
```

```
        Fld2 SMALLINT
) TYPE=InnoDB;
replace `_cet_files_`
(`Name`,`Type`,`Reclen`,`Keylen`,`Table`,`Key`,`Record`)
  values
('example','I','200','20','example','`Key`','`Fld1`,`Fld2`');
load data infile 'c:/tmp/example.txt' into table example
  fields terminated by ',' enclosed by '"' lines terminated by
'\r\n';
```

As you can see, the utility says that there are no inconsistencies in this part of the input file and creates SQL statements to create the SQL table and import the data. Bsqlconv Run again and redirect its output into a file:

```
bsqlconv example -t Record1 >example1.sql
```

This will create a file in a format compatible example1.sql the mysql utility. You will need this file later.

Now we have to analyze the rest of the input file. Bsqlconv run again using the key 'Record1' as the beginning of the analysis:

```
bsqlconv example -f Record1

Input file: example
From: Record1
# Indexed file 'example' Key Length 20 Record Length 200
No inconsistencies found
drop table if exists example;
create table example
 (
        Key CHAR (20) BINARY NOT NULL PRIMARY KEY,
        Fld1 CHAR (15) BINARY,
        Fld2 SMALLINT,
        Fld3 FLOAT
) TYPE=InnoDB;
replace `_cet_files_`
(`Name`,`Type`,`Reclen`,`Keylen`,`Table`,`Key`,`Record`)
  values
('example','I','200','20','example','`Key`','`Fld1`,`Fld2`,`Fld3`')
;
load data infile 'c:/tmp/example.txt' into table example
  fields terminated by ',' enclosed by '"' lines terminated by
'\r\n';
```

As there were no inconsistencies, we can save the SQL statements in another text file:

```
bsqlconv example -f Record1 >example2.sql
```

Now we have finished analyzing the file containing records and found two formats. Therefore will require two MySQL tables to move data from this file to the MySQL database. We created two SQL scripts to create these tables and to import the data. However, both scripts use the same file name and table - 'example'. Open the script file with any text editor, and replace all 'example' for 'example1' for example, in the file and example1.sql 'example2'dentro example2.sql file. You can also replace the field names by others more meaningful. For example, if the first field is used to record the customer's name, replace 'Fld1' for 'CustomerName' etc. Be careful while editing files do not change, as applicable, a single quote, double, and inverse.

2.9.19 Exporting indexed ISAM data files to text files

The utility expanded blist is used to export data from indexed files to text files isam. The new option-e in the command line utility Blist, makes creating a file with a format to be imported into the MySQL database with the LOAD DATA statement.

Remember that you need to create two files, using the key 'Record1' as cutoff:

```
blist example –e –t Record1 >example1.txt
blist example –e –f Record1 >example2.txt
```

2.9.20 Creating MySQL tables and data imports

First you must move the text files of the exported data to the computer where the MySQL server. You should put the files in the location used by the LOAD statement within files. Sql. Bsqlconv Default utility uses the C :/ tmp but you can change it to whatever you suits more to your setup.

Start the utility mysql as root and put w32 as the default database. If all parameters are correct and the MySQL server is running you should see below:

```
mysql --host=<host> --user=root --password=<pw> w32

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 13 to server version: 4.0.10-gamma-max

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Now run the scipt example1.sql:

```
mysql> source example1.sql
Query OK, 0 rows affected  (0.03 sec)
Query OK, 0 rows affected  (0.06 sec)
Query OK, 1 rows affected  (0.00 sec)
Query OK, 2 rows affected  (0.00 sec)
Records: 2  Deleted: 0  Skipped: 0  Warnings: 0
```

The utility runs mysql script and display the results list. When four messages OK means that the table was created and the data were imported into the database. Now you can run the script example2.sql and import the rest of the data file:

```
mysql> source example2.sql
Query OK, 0 rows affected  (0.00 sec)
Query OK, 0 rows affected  (0.05 sec)
Query OK, 1 row affected  (0.00 sec)
Query OK, 9 rows affected  (0.00 sec)
Records: 9  Deleted: 0  Skipped: 0  Warnings: 0
```

Now you have all the original ISAM data file transferred to the MySQL database next step will be to modify the sources CET BASIC using the.

2.9.21 Modifing CET BASIC programs

Note that changing CET BASIC programs will only be required if the ISAM indexed file, split into two or more tables MySQL If the file format is homogeneous isam file may be transferred to a single MySQL table, and you will not need any modifications source programs.

2.10 CET W/32 Library

The power and flexibility of the CET BASIC can be increased by using the library functions Management Development within its own 'app'. These libraries contain a wide variety of functions that can be linked into your program to execute operations that would be difficult or impossible to program using BASIC only.

2.10.1 The POS windows library

The software has been implemented CET Window System Phase One as a library that can be called from programs w/32. This program is offered for compatibility with DOS and UNIX product.
If you are going through windows programs with existing POS, we suggest you replace your windows for text-based dialog boxes, designed with the dialog editor, as a first step to creating Windows applications. You can also use the cChoiceList and Case w/32 MSGBOX to give your application a quick look windows.

2.10.2 The bar code CET library

- The Library Bar Code CET provides the functions required to generate and print barcode from CET BASIC program. This is done by using a simple CALL statement to Bbarcode function.
- With Barcode Library CET, can generate any of the following barcode types:
- • UPC-A and E
- • EAN / JAN 8 and 13
- • 2 and 5 digit supplemental
- • Interleaved 2 of 5
- • Extended Code 39 and 39
- • Codabar
- • Code 128 and UCC-128
- • MSI Plessey Standard
- • Extended Code 93 and 93
- • Zip +4
- • Postnet
- The function returns a string Bbarcode produce the correct bar code to send to one of the following printers:
- • Postscript
- • Epson 9 pin
- • IBM Proprinter 9 pin
- • HP Laserjet at 150 dpi
- • HP Laserjet at 300 dpi
- • Epson 24 pin
- • 24 pin Toshiba
- • Standard Okidata Microline
- • IBM Proprinter 24 pin
- • HP Deskjet 150 dpi
- • HP Deskjet 300 dpi
- • HP Paintjet 180 dpi
- • HP Laserjet 150 dpi Compat Mode
- • HP Laserjet 300 dpi Compat Mode

2.10.3 The file access library.

CET BASIC can be linked (linked) with C language routines following paragraphs CET Guide Interface with C in the User Manual.
Since CET BASIC data files do not use native access methods that run through the operating system, it is difficult to write subroutines implementing operations with files. The File Access Library CET provides functions equivalent to CET BASIC statements OPEN, CLOSE, READ, WRITE and UNLOCK.
Using these function libraries ensures that file accesses are executed through synchronizing with external subroutines C CET BASIC programs callers.
All locks files and records are executed through the library of functions and subroutines that can detect external conditions or end of file locking. The functions used have the following syntax:
int fclose (int channel)
int fwrite (int channel, char * buffer, int lng)
long ftell (int channel)
void fseek (int channel, long offset, int base)
int fread (int channel, char * buffer, int lng)
void finfo (char * nombrefi, struct sfinfo * fi)
char * fgets (char * buf, int max, int channel)
READP int (int channel, char * key, char * buffer)
readn int (int channel, char * key, char * buffer)
readk int (int channel, char * key, char * buffer)
void unlock (int channel)
The File Access Library CET consists of at least two separate libraries: one library is designed to bind to the C key routines and another for binding with CET BASIC programs. You need the Microsoft C compiler to use these functions.

2.10.4 Future versions

CET Software has decided to develop additional libraries within each version w/32 CET. The libraries will extend the power of Application Development System w/32 using routines to support the needs of the developer of management applications.

Chapter 3 The W/32 Compiler

3.1 Introduction.

W/32 The Application Builder is a set of compilers, libraries and utilities that aims to develop management applications in 32 bits. This chapter explains how the compiler and runtime system can modify its behavior with a set of flags and environment variables.
Note that all compiler features explained in this chapter can be used from the command line. But we recommend that if you are developing under Windows 95 or NT, use the integrated development environment to create, debug and maintain their applications. (See other chapters for more detail).

3.2 The W/32 components.

Development System w/32 CET Compilation includes OBW32 administrator, the compiler (OBW32), the runtime system (RUNW32.exe), and resource files (cetwin.obj and cetwin.res).
This product is equipped with a number of libraries. Some libraries provide CET BASIC language support and development functions. Others provide exotic features such as ODBC. The compiler will recognize and seek w/32 standard libraries automatically and ensure that it is believed (or linke) correctly a single executable.
Resource files are a powerful development system which allows you to deploy Windows features in their applications without having to modify its source code. (See Chapter Manager Windows, for details on how to use resource files and chapter w/32 Window Functions for more information on how to use specific features)
The Application Development System w/32 comprises two compilers. Both produce 32-bit executables, one is a 16-bit compiler (BB16) designed for systems development under Windows and the other is a 32-bit compiler (bb32) for use under Windows NT and Windows environments 95. It will use a compiler flag (-32) to OBW32 use the 32-bit version.
If you have used a UNIX compiler CET, will be familiar with the advantages of a 32-bit compiler. On the other hand if you used MS-DOS compiler, will have encountered the problem of not having space for data or source code in some of its extensive programs.
W/32 The compiler supports 32-bit 4GB address space continuum and a lot of space for text and data beyond 4GB. Technically, the size of a one-dimensional table is unlimited and could resized up to 200,000 items. The only limitation (in addition to the maximum of 4GB) is the combined size of real and virtual memory (swap file SWAP). For example, suppose we have 16Mb of memory, and try to run a program that requires 20MB. In this case, the unused programs and data parts less recent use in your program will be flushed to disk. This does not affect efficiency, means you can run programs larger than the available memory on your computer.

3.3 Creating W/32 programs

The compiler accepts programs w/32 sources that have been saved in ASCII format, as these are easily ported between different operating systems. The THEOS Compatibility Guide specifies the information needed to carry BASIC source files from that environment.
To import existing programs in BASIC CET, simply copy the source files into the desired directory. If the source code is in a MS DOS with Windows built, no need to change the files before recompiling.
W/32 new programs can create using any text editor, but we recommend using the integrated development intone, which has been designed for that purpose. For more specific information, see the Language Reference Manual CET BASIC.
For now, it is only important to know that BASIC programs you create must have a. B. Case lines need not be numbered. If you want to practice the features that are explained throughout this chapter, we suggest that you use a text editor and create a simple program called tst.b that contains the following line:
print "hello world"

3.4 Compiling a program with W/32

OBW32 Compilation Manager is a program that calls the compiler multipurpose (default BB16), links (linka) program with the required libraries and creates an executable Windows 32-bit. Exe. There is a flag available to invoke the 32-bit compiler to use under Windows 95 or NT.
OBW32 operates similarly to the DC, the C language compiler UNIX This is the syntax supported and maintains compatibility between products CET BASIC for Windows, DOS and UNIX.

OBW32 syntax shown below. Note that brackets indicate that an item is optional. The names of program source files and / or modules are objects to indicate italics are variable and must be entered by the operator. Names can be in any order and in case sensitive. Windows does not distinguish file names are case sensitive.
OBW32 [-flags] [pgm] [B-pgm] [C-PGMs] [A-PGMs] [O-PGMs]
-Flags
is any valid combination of options or flags used to control the operations of the compiler. Enter OBW32 no parameters and show the supported options and their role on the screen.
The following example will use the 32-bit compiler to create an executable file named pgmnuevo and with the default extension. Exe. Note that the executable program name is followed immediately by the-o option.
Pgmnuevo Obw32-or pgmnuevo.b
pgm
Is the name of the executable program. If not specified, use the name OBW32 cetwin.exe default. For example, the following command will use the 16-bit compiler to produce the executable cetwin.exe default.
OBW32 pgmnuevo.b
B-pgm
is the name of the source file to compile BASIC. Must have ASCII format with a. B.
C-PGMs
is the name of one or more programs in C language compiled. To use this feature, the program should have a. C. The Microsoft C compiler version 8.0 should be available on your system.
Note that w/32 includes a variety of functions C. These functions are explained in a chapter of this manual. Other available functions are explained in the Manual CET BASIC Libraries
A-PGMs
is the name of one or more programs in assembly with the. asm. These programs should be Microsoft MASM. (See the description of the compile-S)
O-PGMs
is the name of one or more object modules with a. obj.

3.5 The compilations 'flags'

Compilation options control the execution of OBW32. Enter the command with no options to get the list of options and environment variables supported. Each build option will be described in this section. It is important to remember that the compilation flags must be entered exactly as they appear here, either uppercase or lowercase.

3.5.1 32 bits Option (-32)

The -32 option should only be used under Windows NT and Windows 95 that will be used to specify the 32-bit compiler. If this option is omitted, the compiler will use 16-bit by default. example:

```
OBW32 -32 -o pgmnuevo pgmnuevo.b
```

3.5.2 Generate assembly code as output (-S)

The-S can be used to inform the compiler generate ends after the assembly language program. For example, the following command generates compiled pgmnuevo.by end when pgmnuevo.asm within the current directory.
OBW32-S pgmnuevo.b
Assembler Program output format is Microsoft MASM.

3.5.3 Define a symbol (-D*símbolo*)

When your BASIC program contains a C preprocessor statement (such as # ifdef) this option is used for a given value the preprocessor symbol. (This feature is explained separately in the Language Reference Manual CET BASIC.)

3.5.4 Select line error (-#)

The - # can be used to specify how the error messages displayed in execution. When used, displayed ERL current line number within the source file that caused the error. In any case, the default is to display the number of the last line that was successful.
This option is particularly handy when a BASIC program contains statements with and without line numbers. For example, the following code with ERL = 120 end if - # not included. Otherwise ERL will be equal to 2.

```
100    X = 0 \ GOTO 120
LÍNEAMAL: PRINT 1 / X
120    GOTO LÍNEAMAL
```

3.5.5 Float text editor (-Zi)

This feature can be used to pass text-based programs to Windows programs with little or no modification of BASIC code. (note that this feature can not be used under Windows for Workgroups).
This flag must be specified at compile time with the 32-bit flag:

```
-32  -Zi
```

Example:

```
OBW32 -32 -Zi -o miprg miprg.b
```

When you run the program you will see a white box with a black edge every time you reach a judgment LINPUT, LINPUT USING, OR GET. The box disappears (leaving the data on the screen) when the operator presses the ENTER key.
This box is a Windows control type, therefore, you can use the mouse to position the cursor within it, and to select characters that fence to clear. Pressing TAB will automatically select all the characters. In fact, the edit control will operate as a kind application windows except the Home and End keys not reposition the cursor. (But if INP values recognize these keys).
The width of the edit box is determined using the mask USING. If your program uses LINPUT sentences without USING, then you will see a box of 16 characters by default. And if the data is greater than the editor appear a horizontal scroll bar. (This bar does not appear in a sentence LINPUT USING).
There is a function to set the default editing controls cLinputOpts call. This function has no effect on programs or LINK CHAIN chained.
The syntax of the function is :

CALL cLinputOpts (long%, use3D%, todos%)

Where
• long% is the default length of the edit controls for sentences than LINPUT USING.
• Use3D% Set to 1, the controls are three-dimensional. Note that these controls have a height greater than one line and thus overlap the text which is next.
• All% Set to 1 selects the entire text to edit if you are using a literal mask. This means that the text will appear in blue when the control.
It is recommended that the variables of background color and foreground have these values to get the colors of the windows, the file can be updated within W32APBLD.w32:

```
B_DFLTFGC=0
B_DFLTBGC=7
```

3.5.6 Define the directory path headers. H (-I*path*)

This option can be used to specify the directory containing the files referenced by the C preprocessor statement # include. The name following the-I option will be passed to the compiler and the compiler w/32 C.
For example, suppose the file pgmnuevo.b contains the # include "getfld", you could use the-I option to tell the compiler that the file is in getfld \ include directory, using the command:
OBW32-I \ include-o pgmnuevo pgmnuevo.b
Note that you do not need the Microsoft C compiler preprocessor directives when using C. (See Manual CET BASIC Language Reference for more information about the C preprocessor).

3.5.7 Include a library on linker (-l*name*)

Normally, the program seeks to link the current working directory bookstores, but first check the directory specified by the environment variable LIB (default c: \ cetlib). Bookseller option (the "L" in lowercase is important) can be used to pass the name of additional libraries containing compiled or assembled functions required by the linker.
For example, LIB Utility Microsoft C has been used to create a library of C functions called myobj.lib. If you make a call (CALL) to these functions within your BASIC program, the compiler directly include this library provided he makes the following:

```
OBW32 -o pgmnuevo pgmnuevo.b -lmyobj
```

3.5.8 Line numering (-Ln)

The line numbering option is used to delete the lines number verification. Otherwise, it will fail if the line number for BASIC statements are out of order or are invalid.

3.5.9 Only compile .obj (-c)

The-c option can use to compile source programs ending after generating an object module. If this option is omitted, the module intermediate object is deleted when the program link (Linke). For example, the following command will end once pgmnuevo.obj generate the file:
OBW32-c pgmnuevo.b
The following command will compile the C program bpgm.b cpgm.cy the BASIC program will end after generating cpgm.obj and bpgm.obj within the current directory:
OBW32-c cpgm.c bpgm.b
If you are going to compile programs in C language, the compiler invokes OBW32 Microsoft C. To use this feature requires version 8.0 (or higher).

3.5.10 Especify the .exe file name (-o nom_ejecutable)

The-o option can be used to create an executable program with a specific name. For example, the following command will compile the BASIC program pgmnuevo.by pgmnuevo.exe produce the executable.
Pgmnuevo OBW32-or pgmnuevo.b
If you omit this option, the default name will be assigned to the executable will cetwin.exe. The following compile pgmnuevo.b as cetwin.exe.
OBW32 pgmnuevo.b

3.5.11 Preprocessor and preprocessor only (-P y -PO)

If the Microsoft C Compiler is within a directory pointed to by the PATH environment variable, all C preprocessor commands can be included such as # define, macro substitution and conditional compilation with # if.
The processor C can be invoked with the-P. If you specify the-PO only run the pre-and the result is stored in a file with the same name as the source file with the extension. I.
This feature also allows variables that can be defined and assigned within the command line as follows:
OBW32-P-DABC-Ddef = xyz pgmnuevo.b
This command is equivalent to entering the following lines at the beginning of your source file pgmnuevo.b:
define abc
define def xyz
The C preprocessor is a special feature that is addressed separately in the Language Reference Manual CET BASIC.

3.5.12 Report (-R*paso*)

The report option is available for debugging purposes when you need to know the exact command line used in each step OBW32. The letters following are the options to display the different steps of OBW32:

-RB compilación BASIC -RC compilación C

-RA Ensamblador -RL enlace

3.5.13 Resource (-r nombre)

The resource option is used to compile a resource file with the specific name nombre.rc modified within a file with the same name and extension. Res. Since only the-r option is used to compile resource files, the extension is assumed. Rc and should not be specified on the command line. For example:
OBW32-r myuser
See chapter Windows Manager for more information on how to use the features of resource files.

3.5.14 Track code -trace- (-T[*rastrea-sub*])

The-T requires OBW32 to include the line number of "trace" when compiling a program containing the ON ERROR statement. For example:
OBW32-T-or miprogrm miprogrm.b
The-t option is also used with the name of a subroutine C. In the following example, misub.c is bound within the BASIC program miprog.exe so that is called automatically after each BASIC statement.
-O-Tmisub OBW32 miprog.b myprog misub.C
Note that this feature is useful if you are compiling your programs from the command line (Under Windows Workgroups). The Integrated Development Environment includes its own source code debugger that can be used to simplify the process.

3.5.15 Ux-basic (-U)

This option is used to indicate that all data files used in the program are opened, read and written in compatibility mode with UX-BASIC. When this mode is specified, the program files written from BASIC w/32 can later be read by UX-BASIC programs and vice versa.
Using this option is equivalent to entering the judgment "OPTION Matio 0" at the beginning of the program and open each file with the option UX.

3.5.16 Version number opction (-V)

The-V option is used to display the version of the compiler and its serial number on the screen. e.g.:

```
OBW32 -V
```

3.5.17 Windows Objects (-wo nombre)

This option is used to suppress the default file w/32 links (or linka) during compilation, called cetwin.obj. If you have a copy that has been modified using the procedures shown in Chapter Framework, then use the file you specified in the option.
In the following example, the first command will create an object file with the name miwin.obj. The second command will link the file (without the. Obj) into the executable test.exe.

```
OBW32 -c miwin.b
OBW32 -o test test.b -wo miwin
```

3.5.18 Windows resources (-wr nombre)

The compiler automatically be linked to the file cetwin.res Framework to generate 32-bit programs with certain characteristics specific to Windows. The file containing these features can be found in a modified resource file called cetuser.rc.
If a copy of the file has been modified cetuser.rc using the procedures of Chapter Framework, then this option can be used to indicate that the file with the specified name should be linked into the executable.
In the following example, the first command will compile a new resource file called miusuario.res. The second link the resource file into the executable tst.exe. Note that the. Rc and. Beef are assumed by default and should be omitted from the command line.

```
OBW32 -r miusuario
OBW32 -wr miusuario -o tst tst.b
```

3.6 Environment variables used by the compiler

Environment variables are used to pass additional information to the compiler or to modify the behavior of the system w/32 execution. These variables should be included in the autoexec.bat file (along with the variable LIB and INCLUDE) to be recognized during the compilation process. Use the following syntax to load an environment variable:

set variable = value

3.6.1 B_CMDFLAGS

Fixed compiler options making them available for each build. For example, if you use the library (optional) Phase One Toolkit, you should enter the library option to tell the compiler where to find these routines.
B_CMDFLAGS SET = "-lPostk"
The following example illustrates how you can select B_CMDFLAGS OBW32 so that files always use the Framework shortlisted located in the current directory. Separate multiple options with a space like this:
B_CMDFLAGS SET = "-r look" "-wo miobj"
This variable will tell the compiler to enable the defaults and display their values. Selecting this variable is no longer needed in the command line options present in the variable.

3.6.2 B_THCONV

Loads 1 to convert string literals to values THEOS. This will ensure that the codes can be converted to international characters (with ASCII values greater than 128) to your assembly output.
SET B_THCONV = 1
Note that the values containing the environment variables are not lost after running the build process if you want to clear an environment variable run:

```
SET VARIABLE =
```

3.7 Environment variables used by the runtime system

A lot of variables can be used to change the operating mode of the system implementation. For example, if you want to port an application from the operating system and wants THEOS w/32 programs operate in the same way, load the following variable:
SET B_EMULATE = THEOS
Variables Execution System, as B_EMULATE be loaded into a special initialization file. When the application starts, the system looks for a file execution with the same name as the application with the. W32 in the Windows directory (usually \ Windows). If the initialization file for a specific application does not exist or can not be found, will be used instead the default file W32APBLD.w32. (Any program chained or bound to always use the same main initialization program that is using the main program.)
The initialization file is editable W32APBLD.w32, is a text file that loads all environment variables to null or empty strings, so as to use the defaults. Edit the file (or preferably a copy) to load any variable to a specific value. Its content should not be replaced when installing a software update.
For a list of available environment variables, run OBW32 without any parameters. These appear grouped by function.

3.7.1 Variables related to the compiler and runtime system

These variables indicate the location of the compiler, runtime system and its components, thus avoiding having to update the autoexec.bat file.

3.7.2 B_CETBIN

Indicates the location w/32 compiler. By default this variable is updated::
B_CETBIN=C:\CETBIN

3.7.3 B_CETINC

Indicates the location of the runtime components cet. Default is current directory cetlib \ INC:
B_CETINC=C:\CETLIB\INC

3.7.4 B_CETLIB

This variable indicates the location of components cet win. Default is current directory c: \ cetlib.
B_CETLIB=C:\CETLIB

3.7.5 B_BRTMDIR

This variable indicates the location of the system running WIN CET. Default is current directory c: \ windows \ system:
B_BRTMDIR=C:\WINDOWS\SYSTEM

3.7.6 Environment Variables related files and input / output

3.7.7 B_CSHFCLOSE

Load this variable to 1 to close all files before running a sentence CSH / CSI. This variable is subject to the default which is not close open files.

3.7.8 B_FPATH

Load this variable to the list of directories where different locations will pick the OPEN statement before launching a file not found error. B_FPATH supports a string of up to 255 characters. Separate each path with a semicolon symbol (;). For example, if the file "question" to be opened. BASIC first searches the current directory. If this file is not in this directory and loaded B_FPATH variable as below, then the program will try to open the file "D: \ FPA \ question".

```
B_FPATH= D:\APF
```

Note that if the variable is loaded B_FPATH search will be conducted for all open files except SEQUENTIAL OUTPUT mode. In this case, the runtime system will always look for the file in the current directory.

3.7.9 B_LIBFNTYP

Load this variable to 1 so that the file name in the format THEOS (abc) are open as \ b \ a \ c. If this variable is not set to 1 BASIC will try to open the file as. \ A \ b \ c.

3.7.10 B_LKPATH

Load this variable search paths used by executable files called by the judgments LINK, CHAIN, and RUN. . Separate each path with a semicolon symbol (;). Note that behaves like Windows w/32 to find an executable file. The search sequence is:

- The directory where the calling program is executed.
- The current working directory
- The \ Windows or \ Windows \ system
- The directories set in the PATH
- Directories within the variable set B_LKPATH

3.7.11 B_REOPENS

Tells the OPEN statement the number of retries to access an indexed file. By default only try it once.

3.7.12 Variables affecting the use and control of printing

3.7.13 B_FFPTR

If set to one, it shall salo page when you close the printer. The default is not to do the page break.

3.7.14 B_MAPPRTn

Select the name of the file containing the control sequences for a specific printer n, where n is the number of printer. The following example shows that the escape sequences that functions with PRINTER1 CRT is used in the file hplaser.lpr:

B_MAPPRT1 = HPLASER.LPR

Note: Printing under Windows it is in implementation notes in the appendices of this manual. For more information on using B_MAPPRTn, see chapter formatted output in the Language Reference Manual CET BASIC.

3.7.15 B_PRINTERn

Check this variable with a specific printer as the Windows printing system in use (B_WINPRINT = 1). You specify a particular printer by:

B_PRINTERn = printer name

Under Windows NT and 95, this name should be assigned to the printer's "Print Manager". Under Windows, you have no option to give a name to a printer, so use the name of the driver installed on the port (for example, "HP LaserJet III").

Note that the only case where the printer number from 1 to 9 in the name of the variable is associated with the LPT port for that printer so that you can continue using the feature B_MAPPRTn. If the variable has the value DEFAULT B_PRINTERn or DIALOG, there is no association between the number of printer and LPT port in any Windows environment.

The variable can also be enabled to make the screen a print dialog that allows the user to select a printer from the list specified in the "Print Manager".

B_PRINTERn = DIALOG

Since at present still be printing ASCII text, there will be some print dialog options as "copies" and the selection of specific numbers of pages that are disabled. (To be implemented when there is genuine Windows printing pictures in next releases).

It will use the system default printer if B_PRINTERn is equal to:

B_PRINTERx = DEFAULT

Print is described in detail in the Implementation Notes in Appendix.

3.7.16 B_WINPRINT

Set the value to 1 if you want to use the new printer interface described in B_PRINTERn. B_WINPRINT If the variable is not set, then use the print method type. "Bat". Both methods are described in Notes Implementation.

B_WINPRINT = 1

3.7.17 Variables affecting the screen

3.7.18 B_DFLTBGC

Load the value of the default background color, use a program that when executed.

During the initialization process, the Framework will verify the value of this variable in W32APBLD.w32 file. If this variable is not loaded, l see a black background, before any color you use your program. If you use a background color in your program, we recommend loading this variable with the same color, this will prevent discoloration often undesirable. If you do not use color in your program, the gray will give a totally Windows. This can be done by setting the variables of foreground and background to black and white, ie the values 0 and 7 respectively.
To display a gray background:
SET B_DFLTBGC = 7
For more details on using color, see COLOR statement in Chapter Features Color and Line Drawing Manual CET BASIC Language Reference.

3.7.19 B_DFLTFGC

Load the value of the foreground color, for the same reason as discussed in the previous paragraph we recommend loading this variable with the same value you use in your application program. For example, the following variable will cause the foreground color is black.

B_DFLTFGC SET = 0

For more details on using color, see COLOR statement in Chapter Features Color and Line Drawing Manual CET BASIC Language Reference.

3.7.20 B_SCRNOLINES

Load the number of lines to be displayed on screen. The default is 25 unless B_EMULATE loaded in this case will be 24.

3.7.21 Variables that provide support THEOS

The use of the following variables will allow their w/32 programs run as if they were running under the operating system THEOS.

3.7.22 B_<código-driver>DRIVE

Turn on to affect the way we interpret the physical drive THEOS driver. For example:
SET B_SDRIVE = C: \ TREND
The following program can be used to access c: \ trend \ dms \ data \ dms01:
$ lib = "DMS"
$ drive = "S"
open1: LIB $ & ". DATA.DMS01:" & DRIVE $, direct update
Activating these variables change the default behavior is: first find the file in the current directory, then the path specified in B_FPATH.

3.7.23 B_DOSTAB

Select to LINPUT USING statements that return a value of 9 for the INP TAB character as low THEOS ago. Otherwise, the default is to ignore the TAB.

3.7.24 B_EMULATE

Loading this variable to 1 mimics several features of THEOS, automatically activating the emulation-related variables such as B_FFPTR, B_GINVERT, B_OPTLOCK, B_THCONV, B_THLOCK, B_THPON and B_THVAL.
Enabling emulation attributes THEOS will screen control function as in THEOS THEOS file formats are recognized, that INP values are as expected, and that the escape character will become 27.
Note that division by zero errors will be detected. This is an illegal operation (in C) for good reason: if you were allowed the program may produce incorrect data without any notification would.

3.7.25 B_OPTLOCK

When set to one, will cause the READNEXT, and function as in READPREV THEOS after a RESUME. (These sentences will run without moving the pointer to the file, registry retrying reading). If a record is locked, and this variable is not set, CET BASIC will read the next record, or earlier when running a RESUME.

3.7.26 B_THLOCK

Enabling this variable to 1 the program is suspended when it detects a stall condition. CET BASIC suspends execution only when faced with a judgment or ON ERROR ON LOCK.

If this variable is set, execution of an ON ERROR statement not generate an error when run blocking access statements that direct or indexed files. Refer to chapter "system error handling" the Language Reference Manual.

3.7.27 B_THPON

Activate this variable to one to reverse the default meaning PON / POFF as THEOS. CET BASIC normally shows low-intensity text under MS DOS and Windows environments. Jogging B_THPON ensure the variable use of high-intensity attribute in the foreground.

3.7.28 B_THVAL

Check this one variable to VAL functions and work as THEOS NBR. This will make judgments as having below zero:
PRINT VAL ("abc")
If not used the environment variable, this statement would generate a runtime error. See the Language Reference Manual CET BASIC to see the standard way of dealing CET VAL functions and NBR.

3.7.29 Generic Environment Variables

3.7.30 B_4DYEAR

Load this variable to 1, so that the DATE functions and work with 4-digit DATE.

3.7.31 B_DATEFORM

Charge 2 (European), or 3 (International) to change the default date format, showing dates in American format (MM / DD / YY). The CET BASIC DATEFORM option is used to change the date format in the program.

3.7.32 B_LANGUAGE

Introduces two 2-characters that specify the language, to force the function $ YESNO generate the appropriate text for the languages French (FR), German (GE), Swiss (SW), Finnish (FI), Spanish (SP) , Portuguese (PO) or Italian (IT). The default is English (EN)..

3.7.33 B_USER

Check this variable to 1, 2, 3 .. autoexec.bat in each single user to specify a network system.
This feature is most useful when a program needs to create a temporary file in a multiuser environment. The following code segment illustrates how to create an indexed file with a unique name for each user. The file is stored in the \ tmp Z driver from the remote server.

```
Bgetenv call (addrof ($ user), "b_user")
nomfich $ = "Z :/ TMP / WORK" & $ user
CSI ("DEL" & nomfich $)
CSI ("CREATE" & nomfich $ & "INDEX keylen RECLEN 100 10")
OPEN # 1: $ nomfich, update indexed
```

This will work similarly to a sequential file, in this case, instead of using the CREATE, use OPEN with the option to write the file OUTPUT, or EXTEND option to concatenate the file.

3.7.34 B_ODBCERR

Put the value one, to help you get on screen error messages that may occur when accessing files through ODBC w/32.

3.7.35 B_YR2000

Check this variable with the last two digits of the year that will be interpreted as 20XX. This feature allows you to have dates of 1900 and 2000 without having to use a four-digit notation.
For example, suppose you have no data prior to January 1, 1925. In that case it could do:
B_YR2000 = 24
This would cause the judgments DAY ("01/01/24") and DAY ("01/01/2024" were identical.
Using this example, the following dates are interpreted as:

```
01/01/00   01/01/2000
31/12/24   31/12/2024

01/01/25   01/01/1925
31/12/99   31/12/1999
```

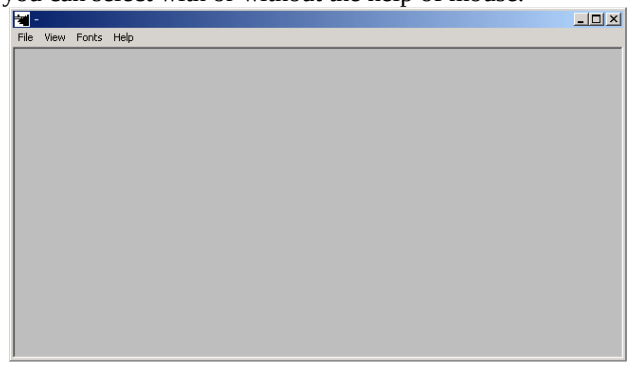
Chapter 4 The framework

4.1 Introduction

When you create a program using the Application Development System w/32 CET, read a set of editable files to determine how to open and manage the window showing its application. These files will be called the Framework Windows.
This chapter will cover the characteristics of the framework that can be used without having to modify its source code. If you have not already, we recommend that you compile a simple program that displays some text just so he can go using each feature we go looking.

4.2 The default windows.

By default, w/32 programs have a main window with the following menu bar: File, View, and Help Sources, you can select with or without the help of mouse.



4.2.1 The file menu

By default the file menu has only one command, Exit. Selecting out during the normal execution of a program equivalent to a keypress interrupt (ALT + F4). Unless an ON ERROR routine or ON INTERRUPT available to intercept the interrupt, a message box appears showing "Interrupted Program" (Program interrupted). Otherwise, when you leave, it will display the message "Exit Application" (Application Output). In both cases, you need to press OK to continue.
The message boxes are commonly used in Windows and will be covered in detail in the next chapter. It is important to note that these special message boxes have been added because the typical Windows programs close the windows before leaving, not allowing you to see the latest things that have been written on the screen. (See cForceExit role in the next chapter if you want to remove this behavior).

4.2.2 The View menu

The View menu has two commands, Toolbar (Toolbar) and Status Bar (State Bar). The toolbars show a series of buttons that are quick ways to access the menu commands. These buttons appear just below the menu. By default, there are two buttons, one for the command "Font Select" (Select Source), and one for "Help-About" (Help About). To design your own buttons and keys to abbreviations of your own command menu function CToolBar see in the next chapter.
The status bar is an area at the bottom of the screen, which is used to display status messages, how are the hints menu commands or toolbar.
The toolbar (toolbar) and the status bar (Status Bar) are enabled-disable commands. Selecting once the bar is activated / deactivated and if selected again deactivates / activates.
Note that the toolbar and the status bar is not normally shown. Later in this chapter, we will show how to modify the Framework to use special features you want to enter ("linkándolas") within your applications. The functions that can be called within a BASIC program will be explained in the next chapter. (There are also Windows w/32 functions used to enable or disable these features within a BASIC program).

4.2.3 The Font menu

The Source Select menu allows you to change the specifications of the type, style and size of characters displayed on a Windows box. The default font is a 9-point Courier. (Point size refers to the size of individual characters).

The size of the font that you can use depend on the resolution of your monitor. If you are working in high-resolution (eg 1024x768) can use a power greater than 9 points. If you are working with a standard VGA 640x480 resolution, then the largest font that can display is 9 points. (With the function cDisplayInfo may know the resolution of your monitor from within your program.) Note that you can resize the window, but the maximum area that can be displayed will never be higher than necessary to display 80x25 characters to font size current. If you make the window smaller, will "scroll bar" so you can access parts of the screen that are not.
Although from the menu you can choose any available source, can prevent the operator do this by selecting the source from within the application. (This procedure will be covered later in this chapter.)
If you want to prevent the user to change the font of your program, you can control it from within your program. This feature will be explained later.
Note that the above information applies only to character-based screens created by PRINT statements. In fact you can not change the font in a dialog created with the Dialog editor w/32.

4.2.4 The Help menu

The command "Help About" default displays a message box with information about the version and Property Rights System w/32 CET Application Development. later in this chapter will be an example of how to modify the text to display information about your program.

4.3 The framework default windows

Each program uses w/32 characteristics and default initial conditions that can bind OBW32 (linkar) and cetwin.res cetwin.obj files during the build process. These files are stored in the \ cetlib unless the installation process you specify a different one.
Cetwin.res is the resource file. This contains the code for the menus and their items appear in your compiled programs. The source code of the resource file can be edited, its name is cetuser.rc.
The source code is in cetwin.b cetwin.obj. This file contains a series of comment lines to make it easier to change, if you want to: a) change the default form of work removing the toolbar and status, b) selecting a specific source and / or add other menu commands within their programs.
If cetwin.b list, you will see that all operations are conducted through BASIC subroutines. Each subroutine begins with a sentence $ SUB defining the name of the subroutine. The first of these is called cetPreInit since before the first line w/32 program is executed.
$ EXIT defines the end of the subroutine and acts as a RETURN statement. $ GLOBAL is used to specify the variables that will be shared with other program modules. (In cetwin.b, CmdID% is passed as a global variable.)
After compiling a BASIC subroutine can bind (linkarla) under its program 'main' to produce an executable. The ability to call subroutines BASIC is a feature that will be worn out later in this manual.

4.4 Changing the default framework

The framework is designed to quickly deploy Windows features, you want to use within your application.
To alter the default way of working, first copy the files you want to change within a separate directory with different name. For example, if your files are in w/32 \ cetlib and is \ tst:
copy \ cetlib \ cetwin.b \ tst \ miven.b
copy \ cetlib \ cetuser.rc \ tst \ miusua.rc
Note that the names of the files in the local directory must be different from \ cetlib. Later, probably decide to have multiple copies of these files for each configuration you want. For example, different menus for each of its major programs. The compiler will allow you to easily work w/32 this way.
After making any changes to those that appear in this chapter, use OBW32 to recompile resource files. For example:
Create OBW32-r miusua miusua.res
Create OBW32-t miven miven.obj
Tell the OBW32 command, which is used for the compilation of their programs, how to use Framework, likándolo into the executable. You must use the-wo to replace the default file cetwin.obj by choice. Use the-wr to replace the default file cetwin.res. by
eg
OBW32-o tst tst.b Use cetwin.res and cetwin.obj
OBW32-o tst-wr miusua tst.b Use miusua.res and cetwin.obj
OBW32-wo-o tst miven tst.b Use cetwin.res and miven.obj
OBW32 tst-wo-o-wr miven miusua tst.b Use miusua.res and miven.obj

In the following sections, we will discuss how to delete the default fonts menu and select the desired source. We will also see the use of toolbars and menu bars changed at your convenience, without having to change the source programs.

4.5 Selecting fonts

This section shows how to use the routine cSelectFont to choose the source from within your application, rather than choose from the font menu.
The selectable sources vary depending on the ANSI fixed sources you have your system. Therefore, it is possible to document all parameters necessary to select different fonts. (W/32 must use fixed-width fonts since the program must be able to determine where and how to display a character, the character position multiplied by its width).
If you plan to use other sources other than the default Courier 9 or Terminal must cFontValues routine use for the parameters you need. To use this feature compile a program like the one shown below:

```
PRINT "Enter anything you want to see"
PRINT "and how many lines of the text"
LINPUT A $
CALL cFontValues
END
```

Run the program once compiled. When the program is waiting for input, choose the Select Sources within the menu. The Font dialog box will display the available fonts, styles and sizes. Choose one of the sources, and press ENTER to respond to the judgment LINPUT.
Calling cFontValues will show all the parameters that need to happen anytime cSelectFont change the font type from within your application. (See chapter on Windows w/32 Functions for more information about these routines).

4.5.1 Delete the Font menu

If you plan to use the same type of presentation Windows in all or part of your application, we recommend using the following procedures to modify a copy of the resource files instead of modifying each of its programs. For example, to remove the font menu:
1. Copy the source code of the resource file in your working directory with a different name. For example, if your files are in w/32 \ cetlib and local directory is \ tst:

```
copy \ cetlib \ cetuser.rc \ tst \ miusua.rc
```

2. Modify the code in miusua.rc blurring the lines showing the menu:

```
POPUP "Fonts"
BEGIN
MENUITEM "Select", ID_FONTS_SELECT
END
```

3. Compile a new resource file called miusua.res using modifications to miusua.rc:

```
OBW32-r miusua
```

4. Rebuild your application using this resource file (not the default cetwin.res) specifying the name and code-wr as follows:

```
OBW32-wr-o tst miusua tst.b
```

From now on, the menu font does not appear in any BASIC program to compile the file using miusua.rc.

4.5.2 Choosing the sources within the Windows Administrator

The file contains the subroutine cetwin.obj cetPreInit that is called when initializing Windows Manager. Any statements in this subroutine will be executed before the first line of code in your program run.
If you plan to use a particular font in your application, we suggest you change this file to select the source and stop its program files as they are.
Modify the file cetwin.obj:
1. Copy the source code in the directory cetwin.b working with a different name.

```
copy \ cetlib \ cetwin.b \ tst \ miven.b
```

2. Modify the copy of cetwin.b to use the Terminal font point-9, which is the default. This is done by removing the comment to the next statement in the subroutine cetPreInit the beginning of the file:

```
REM call cSelectFont ("Terminal", -12, 400, 0, 255, 49)
```

If you want to use another source, enter the display settings with a call to cFontValues (See the procedure explained in the previous section).
3. Compile the modified program:

```
OBW32-c miven.b
```

Now, compile any program in the \ tst can be linked to the new file. Obj with the following command. The specified font will be used automatically.

```
OBW32 -wo miven -o tst tst.b
```

4.5.3 Displaying the default toolbar

You can also modify the Windows Manager to automatically display a toolbar. The default toolbar consists of two buttons. The first of these may be used
as a quick way to display the Source Select menu. The second button (the?) Is quick access to Help About command (Help-About).
To display the toolbar, change the cetPreInit subroutine in file copy cetwin.b removing the comment (REM) of the following statement (and recompile):
REM call cShowToolbar (1)
When you call this routine will automatically make the toolbar reappear, which was hidden, when you run any of the programs compiled with the modified file cetwin.b. If you set a new toolbar with CToolBar routine (see section Functions Window) will automatically appear. Otherwise, will be used the default toolbar.

4.5.4 Viewing the status bar by default

The default status bar is a line at the bottom of the window that displays the status of the program in progress ("running" and "wait for input") along with specific messages of the elements of
menu. The subroutine cetwin.b cetPreInit in the file contains the code that activates the status line. To implement this feature, remove the following line commented out (and recompile):
REM call cShowStatusBar (1)
Calling this routine will automatically display the status bar when you run any of its programs compiled W32. You can call a special function, the cStatusText from its program to replace the default messages from other messages that you choose. Please refer to the chapter Functions Window to see in detail how to use this feature.

4.5.5 Changing the default title in the main window

By default, no title will not appear in the main window. To define a function using cAppTitle title:
cAppTitle call (ABC Payroll System)
You should be aware that the text chosen not exceed the width of the window, because it exceeded the characters will be truncated.
CAppTitle function should be used within a BASIC program. If used within the subroutine cetPreInit cetwin.b contained in the file (and recompiling), then any program that linke with this. Obj has the same title.

4.5.6 Changing the default menu bar

CET w/32 App Builder gives you many options to incorporate features of Windows within their programs. You can display your classic menus in the main window without having to use the drop-down submenus system. You can also choose to completely replace their existing menus or display your classic menus and give the user the option to use a submenu with or without a mouse.
The Framework is designed to help in the development of Windows submenus. By default, programs w/32 displayed a menu that includes options: File, View, and Help Sources. This menu can be redesigned to display more than two dozen different menus with a maximum of 50 250 menus and menu commands.
For example, suppose you want to run Windows NotePad from the application menu. First, create a copy of cetwin.by cetuser.b using the procedure described in the previous section for selecting a font.
Then, edit the file copy cetuser.rc adding a new menu command to be called 'Notepad' menu in the code:
Selecting the menu command 'Notepad' a submenu with the element 'Run'Añadiendo amspersand symbol' & 'in front of the letters N and R Run Notepad, you create' shortcut keys' that are listed in the menu that appears underlined.

```
POPUP "& Notepad"
BEGIN
MENUITEM "& Run", ID_WINSUB1
END
```

Whenever you want to add a menu item to the end, you must assign a unique identifier, as in the example ID_WINSUB1. Since it can take up to 250 menu commands, identifiers to be appointed from ID_WINSUB1 ID_WINSUB250.
To process a new menu command, modify the SELECT statement in the subroutine-CASE CETWINSUB in BASIC (cetwin.b end of file). By default the CASE statement is not commented out. For example, to add an element called "Notepad" to launch for the name notepad.exe enter:

```
CASE 1
```

CALL cWinExec ("notepad.exe", 3)
The routine cWinExec is specially designed to run MS Windows programs, being explained in the chapter on Windows w/32 Functions.
When the user selects the menu command 'Run', then the Framework will call the subroutine to process cetWINSUB selection. The menu identifier '1 'in ID_WINSUB1 cause the CASE statement runs one of this subroutine.
Note that the file contains a sentence cetwin.b $ GLOBAL, which passes the handle (1 to 250) within the variable CmdID% to your program. If your program needs to use that information, please refer to the chapter of BASIC subroutines for instructions on how to use $ GLOBAL.
After making the changes, you will need to recompile files OBW32. For example:
OBW32-c cetwin.b
OBW32-r miusua
Notepad will now appear in the menu of any program linke with these resource files.
Using the following keys can be set up to 50 different menu bars:
END POPUP MENU BEGIN MENUITEM
The MENU keyword is used to identify a unique menu. The syntax is:

MENU PRELOAD identifier DISCARDABLE

The identifier is a unique value that refers to a specific menu. For example, the default identifier for the menu is IDR_MAINFRAME. Although you can change the default menu, each application must have one. If you want to add other menus use IDR_MENU2 A IDR_MENU50 identifiers.
BEGIN and END keys are used to specify the start and end of a "group". In the case of the MENU key, all content elements are associated with that particular menu.
The POPUP keyword identifies a menu item, and the words BEGIN and END contain definitions of each item of property that popup submenu.
The key identifies a specific group POPUP menu. A maximum level POPUP is a single menu blinds as default File menu. A maximum level menu items can also contain nested popup menus. These are considered separate menu groups are identified by □ at the end of the menu text. When selected, the menu also appear to the right of the menu item, so it is called cascading menu. The syntax for POPUP key is:
POPUP "title"
The title is the name of the specific menu. The quotation marks are required. In the title you can designate one character mnemonic (to be emphasized) that can be introduced to select the item. This precede the character to be used with a amspersand sign (&). For example, the following code will display "File" menu by default with the "F" underlined so that users will know which key to press to select.
POPUP "& File"
The MENUITEM keyword is used to define a menu item to select, just as a group defined POPUP menu. Associated with each menu item is a unique identifier which will be passed to the subroutine cetWINSUB (as in the example Notepad) at the time that the item is selected. The syntax of the key is:
MENUITEM "title", ID
The title is the text that will be displayed in the menu. A specific mnemonic can be associated with the title the same way you do with a degree from POPUP. The unique identifier is a value ranging from ID_WINSUB1 to ID_WINSUB250.
There is a second way to use the key MENUITEM that may be used to display a horizontal black line if you want to group items in a menu.
MENUITEM SEPARATOR
SEPARATOR menu item has been used in the default File menu to draw a line before displaying the item Exit (Exit) on the menu.
The following example defines a menu with three main types popup menus: File, Find Customer and Add Customer. This menu is the default menu and using the identifier IDR_MAINFRAME. Note that ID_APP_EXIT is a special identifier for the option to exit the program.

```
MENU PRELOAD IDR_MAINFRAME DISCARDABLE
BEGIN
POPUP "& File"
BEGIN
MENUITEM SEPARATOR
MENUITEM "E & xit", ID_APP_EXIT
END
POPUP "Find Customer"
BEGIN
MENUITEM "On & name," ID_WINSUB1
MENUITEM "On & phone" ID_WINSUB2
```

END
POPUP "& Add Customer"
BEGIN
MENUITEM "& New", ID_WINSUB3
END
END
Note that the w/32 gives full control in response to an event your Windows program as if it were a menu selection. For example, you may at one time disabling a calling option cEnableItem so that it can not be selected. (The text appears in gray instead of black.) CEnableItem Another call to be executed to allow selection.
During the execution of a program, there may also be times when you want to display a different menu altogether. This can be done with a function call cNewmenu. Please refer to the next chapter for more information on these special functions menu.

4.5.7 Adding specific messages for menu items

A special feature in the resource file allows you to add a short message to describe the purpose of any of the menu items. These messages appear in the status bar when the item is highlighted (it lights).
Messages are defined in a separate file called STRINGTABLE cetuser.rc. This section contains the comment "Add notes here in ID_WINSUB ..." (A comment is a line beginning with two slashes forward '/ /'.). The syntax of the line point is:
identifier "string \ nHelp"

The ID is the unique ID value associated with the menu item ID_WINSUBx. The string is the text to be displayed in the status bar. The aid is a short string (one or two words) that appear in a small window when the cursor is moved to the button of the toolbar associated with the menu item. Note that the string and the help are separated with the newline character (\ n).
The following will define a prompt message for menu Add New-Client which serves as an example to the previous section.
ID_WINSUB3 "Add a new customer to the database \ nCreate"
When the menu item appears highlighted, the text "Add a new customer to the database" appears in the status bar (when visible). If a button on the toolbar has been defined for the menu, timely assistance "Add New" will appear in a window when the mouse cursor is over the button. (The line feed character (\ n) is used to separate these two elements in the chain).

4.5.8 Changing the message About

The menu item "Help-About" (Help-About) displays the application name, version and information of property rights in a message box. By default, this information refers to the Application Development System w/32 CET. Follow these steps to change the menu Help-About and display information from your application.
1. Go to the section in the file menu cetuser.rc and note that the menu POPUP "& Help" displayed MENUITEM "& About Development System w/32 appl ...". Change the text so that it appears the name of your application.
2. The section contains the dialog caption or title for the window and the text lines are displayed in the dialog box called IDD_ABOUTBOX DIALOG. The lines can be modified are:
CAPTION "About Apli...W/32 Development System"
LTEXT "\ STBN w/32 Applied Development. Version 8.5 \ n \ nCopyright \ 251 CET
Software Inc. from 1986 to 1995. \ N All rights reserved.
When you edit this text should be careful. Characters that do not fit in the window will be truncated. Note also how the line feed character "\ n" is used to position the text in the dialog box. "\ 251" is a special character that can be used to display the symbol of property rights (copyright).

4.5.9 Changing the title of a message box w/32

W/32 programs automatically display some system information in a message box as it appears when you exit the program. By default, the title of the box is the name of the program. The title can be changed to make it more meaningful to the user in section STRINGTABLE (String Table) file cetuser.rc. Default lines are:
PRELOAD STRINGTABLE DISCARDABLE
BEGIN
AFX_IDS_APP_TITLE ""
END
The null character can be changed to the name of your application. If you do take care that the text within the window Straighten if not, the excess characters are truncated.

4.5.10 Changing the program icon

By default, all programs CET w/32 icon displayed in the Help About dialog box-In and at the bottom of the screen when a program is minimized. The icon will also be used if you create your own program element in the main program.
The application icon is defined in section cetuser.rc icon file with the following line:
ICON IDR_MAINFRAME DISCARDABLE "CETICON.ICO"
The default icon is stored in the file directory ceticon.ico \ cetlib \ inc if you used the default PATH in the installation process.
To design your own icon, copy the default icon in your local directory with a new name. eg
copy cetlib \ inc \ ceticon.ico \ tst \ miicon.ico
Now, use the program to modify the new Imagedit miicon.ico file. Once done, edit the line in cetuser.rc to use the new icon.

```
IDR_MAINFRAME     ICO DISCARDABLE      "MIICON.ICO"
```

4.5.11 Simulating keyboard input with mouse subroutines.

Text-based applications, DOS or UNIX, typically provide some options (menus) that keep the program waiting for the user to make an entry. The same happens when we work under the Windows environment.
As a developer, you have complete control over how your program responds to events Windows like menu selection or mouse click. For example, if a user uses the mouse to a menu selection, your program will be notified. At that point, you can either move forward in the selection process, ignore the event or save the information to be considered later.
The framework is designed to alert an application when specific mouse events occur. Cetwin.b file contains the BASIC subroutines that can be used to detect mouse events .. The following events may be detected:
* CetRBtnDn and cetLBtnDn * Right and left button down
* CetRBtnUp and cetLBtnUp * Right and left button up
* CetRBtnDCIk and cetLBtnDCIk * Right and left double click button (release the button)
* CetMouseMove * Move mouse

In general, only the routines that detect a double click, or release the button to the right or left will be used, as it is at the moment you release the button when an event occurs. For example, you can "drag" the mouse over the items in a menu without making a selection. When you release the mouse button (the top button event), the item is selected.
The button down and moving routines are included in case you need a level accurate information of the movements of the mouse. For example you can use a call to cGetmouse current coordinates for character-based mouse.
The following examples use w/32 functions to insert characters into the keyboard input queue. If your program is waiting for input (eg. LINPUT a judgment) when the button is released, the statement is executed as if it had used the keyboard.
The first example F1 key inserted into the buffer. Note the use of two-character code. The first code below zero means that a key is extended (the F1 key).

```
$SUB cetLBtnUp
REM Llamada cuando el botón izquierdo es soltado.
REM Inserta la tecla F1 dentro de la cola de entrada.
call cInputQ (0)   'Tecla Extendida  (De Función, cursores, etc ...)
call cInputQ (59)  'F1
$EXIT
```

The following example of the operation of the ENTER key cetRBtnUp inserted into the input buffer as soon as you release the right.

```
$SUB cetRBtnUp
REM  Llamada cuando suelte el botón derecho del ratón.
REM  Introduce la tecla INTRO dentro de la cola de entrada.
call cInputQ (13)
```

You can find a table of characters for input codes in Appendix Manual CET BASIC Language Reference.
The following is an example of how you can modify a copy of your file cetwin.b to add mouse support to your application:

```
$SUB cetLBtnDn
```

```
$GLOBAL MROW%, MCOL%, MOUSEDOWN%, SELANCHOR%
REM             Llamado cuando se pulsa el botón izquierdo
MOUSEDOWN=-1
CALL cGetMouse (addrof (mrow%),addrof (mcol%))
CALL cInputQ (3007) \ rem MOUSE.DOWN%
$EXIT

$SUB cetRBtnDn
REM             Llamado cuando se pulsa el botón derecho
REM             No se utiliza en este ejemplo
$EXIT

$SUB cetLBtnUp
$GLOBAL MROW%, MCOL%, MOUSEDOWN%, SELANCHOR%
REM             Llamado cuando se suelta el botón izquierdo
MOUSEDOWN=0
CALL cGetMouse (addrof (mrow%),addrof (mcol%))
CALL cInputQ (3008) \ rem MOUSE.UP%
$EXIT

$SUB cetRBtnUp
REM             Llamado cuando se suelta el botón izquierdo
REM             No utilizado en este ejemplo
$EXIT

$SUB cetLBtnDClk
$GLOBAL MROW%, MCOL%, MOUSEDOWN%, SELANCHOR%
REM             Llamado cuando se hace doble-click con el botón izquierdo
MOUSEDOWN=0
CALL cGetMouse (addrof (mrow%),addrof (mcol%))
CALL cInputQ (3005) \ rem MOUSE.DBLCLICK%
$EXIT

$SUB cetRBtnDClk
REM             Llamado cuando se hace doble-click con el botón izquierdo
REM             No utilizado en este ejemplo
$EXIT
$SUB cetMouseMove
$GLOBAL MROW%, MCOL%, MOUSEDOWN%, SELANCHOR%
REM             Llamado cuando se mueve el botón izquierdo
if mrow%>0 and mcol%>0 and selanchor%>0 and mousedown%<>0
        CALL cGetMouse (addrof (mrow%),addrof (mcol%))
        if row%<> mrow% or col% <> mcol%
                mrow%=row% \ mcol%=col%
                call cInputQ (3006) \ rem MOUSE.MOVE%
                ifend
        ifend
$EXIT
```

In order to pass the values of the variables $ GLOBAL your BASIC program, you can add the following sentences at the beginning of the program:
$ MAIN
$ GLOBAL MROW% MCOL% MouseDown% SELANCHOR%
Variables MROW% MCOL% MouseDown% SELANCHOR% are used to determine the position of the mouse when the event was detected, and save (in SELANHOR%) so that the cursor can be repositioned after the transaction has taken place chosen . (% variables ROW and COL% indicate the initial position of the entry field)
The main program (one that contains the sentence $ MAIN) also has to define mouse events. Large numbers have been used to make it easier to distinguish between keyboard input and mouse)
MOUSE.DBLCLICK% = 3005
MOUSE.MOVE% = 3006
MOUSE.DOWN% = 3007
MOUSE.UP% = 3008
The mouse support is automatically created in the dialog vantanas or display formats created with the dialog editor. See chapter 'Dialogues Editor w/32' for more information about how to handle mouse events.

4.5.12 Special considerations when using windows

There are some important considerations to keep in mind when using the Windows features.
The menu selection and mouse events can occur at any time during the execution of a program. When a user changes to a menu item, the Framework automatically calls a subroutine cetwin.obj. This is an asynchronous event, similar to a hardware interrupt that can occur while running a line of code. An improper calculation may result if the program is currently accessing a global variable, this change suddenly due to the execution of an event.
We suggest you use the following rules to prevent such problems occurring.
1. Disable (temporarily) any menu item that may interfere with the normal data takes.
2. Use signaling variables (flags), traffic light, which means you can check verification variables $ GLOBAL cetwin.obj and shared with your program have changed. If you have spotted a change, you can copy the shared variable within a single variable to which only you can access your program.
3. Only run code from file to configure cetwin.obj 'state' of its program. But all that is processing the information (data, files, variables, etc ...) it from your application.
4. Never write any information in a file or remove it from the screen cetwin.obj. This will ensure that no information can be updated from two places at once.
5. Otherwise, their programs could redraw the screen when you select a menu item. If you update the screen since his departure cetwin.obj file could mix with their programs.
For example suppose you have the following code within the routine cetMouseMove
cGetMouse call (ADDROF (row%), ADDROF (column%))
cGetCursor call (row%, column%)
print "Hello World"
This code will trace the position 'actual' mouse, move the cursor to that position and draws the string "Hello World". This could cause strange effects if this code is executed while the main program screen out data

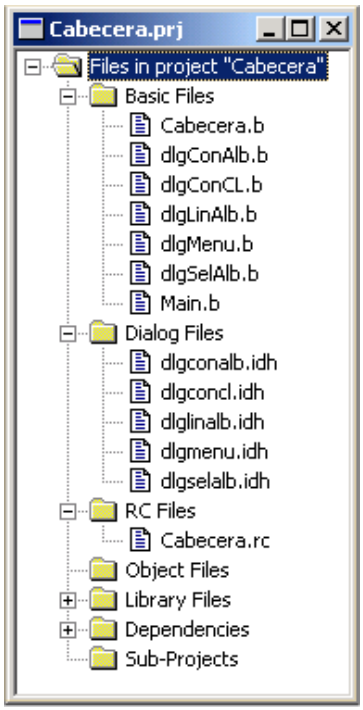
Chapter 5 The W/32 integrated develop environment

5.1 Introduction

The Application Builder w/32 has a graphical development environment, (w32apbld.exe) that has multiple uses for convenient development of their applications. While this program you can create:
• Projects to compile, test and maintain your application executable.
• Source files using a text editor especially designed for writing for long programs. The debugger is also available to expedite the debugging phase.
• User Interfaces (GUI) that consist of a collection of windows that will give you a much more visual for your application.
The information in this chapter is organized into the following sections to help you learn how to create new projects to its existing executable programs.
• Use projects: basic concepts.
• Create projects.
• Add files to project.
• Build and maintain projects.
See the next chapter for more details w/32 debugger and dialog editor w/32
Note for Windows versions 2000 or higher: because of problems with the product W32, W32APBLD not allow proper compilation when it is called from a command window. This problem does not occur under Windows 9X or NT

5.2 Use Project - Basics

W32APBLD you to build projects to simplify the creation, debugging, and maintenance of their applications. You can make use of these benefits with existing applications created from the command line.
When you create a new project, the environment shows a project tree, where each file type is represented as a folder, and hanging from them, such files belonging to the file you created. (For example, the file types are: BASIC, dialogues, Object, etc). This is a convenient way to get an overview of your project.
When you create a project, there are various menu options available so you can build one or more files, or build your project and generate an executable file. W32APBLD keeps the names, location and date of each of the files. Each time you run your project, the program will ask you to recompile the files that have changed.



The basic concept of a set of files that make up a project is the dependency between them. For example, when you create a project that generates mainprog.exe, the program is considered MAINPROG dependent mainprog.b source file, if this source contains an include statement, eg # include "subtot.inc" then also considered which depends MAINPROG subtot.inc file. Both files are subtot.inc mainprog.by dependencies. If any of them is changed, then, then the project will be rebuilt (Rebuilt) to generate a new executable.
There is also time relation between the executable and its dependencies. W32APBLD is considered to be updated when the date (Modified) is after all the source files on which it depends.
W32APBLD uses time and dependency files to determine when a compilation is required. This means that once you have created a project and not have to worry about when to modify a project file. When you go to run the project, W32APBLD detect that there has been a change and will ask you to recompile (Rebuild). If you make a Rebuild (recompile), compile only those files that have changed (the date after the project)

5.3 The program W32APBLD

The integrated development environment w/32 program is invoked by calling cetbin W32APBLD.exe from the directory, or by clicking on the icon CET w/32 group Aplication Builder Windows menu.
After loading W32APBLD program will display the typical File (File), View (View), and Help (FAQ) of Windows program menu. Although it has several toolbars available, most of them are initially disabled.
Multiple menu commands are also available when creating projects, dialogues, or use the debugger. Only the menu commands that refer to projects covered in this chapter.

5.3.1 The Help menu

The command 'Contents' (contained), displays the help system w/32. The help file is for the user manual w/32. If there are new features in the product, the manual and the help file will be updated as soon as possible. Please see the notes of any new information that is not included.
The command 'Help-About' (help - about) shows the version and copyright information W32APBLD.

5.3.2 The View menu

The menu 'see' you have options to show or hide any of the toolbars and the status bar. Most toolbars are designed to work with the dialog editor. If you change the visibility or position of any toolbar, the program will remain in that state the next time it is invoked. You can use the command 'View-Settings', so that all return to their original position.

5.3.3 The File menu

The File menu contains commands initially: New, Open, Find in Files, and Exit Imprt Template Dialog. It also contains the 10 files that were opened more recently.
File-New command displays a dialog box with three options: source (source), dialog (dialogue) and Project (Project). (sources and creating dialogues discussed in other chapters).
File-Open command lets you select a specific path and open a file of the following types:
1. Extension source files. B,. H. Inc, or. Cpu, which are used for originally created files CONTROL Plus THEOS under the operating system. There is a text editor to load and create these types of files. See Chapter w/32 Debugger for details.
2. Files extension talks. HDI. Note that if you open a file. Lb. The program will load the source file into the text editor W/32.l
3. Files of extension projects. Prj.
4. All files of any extension. The program will attempt to load the files from any other extension within the text editor w/32.
Since the command 'Find in Files' has been created to aid in debugging, this feature will be explained in chapter w/32 Debugger.
File-import command Dialog Template is a special feature that helps you to convert text-based displays dialog boxes. This feature is explained in Chapter dialog editor.
Other menu command will appear as you work with the environment, for example, the command 'Window' provides the ability to organize multiple windows you have open at once in a cascade, or quadrants, or to toggle between other, clicking on the name of the same.
Menu specific commands are explained in this chapter, in the appropriate section.

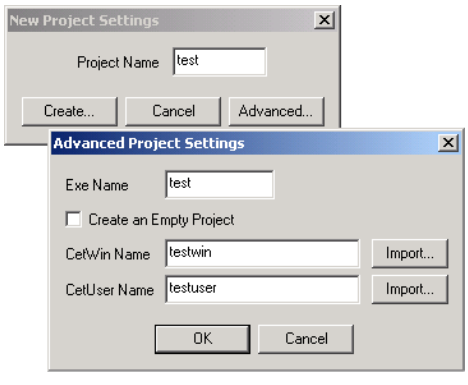
5.3.4 Command Arguments W32APBLD

You can enter the environment and have opened a specific file without having to select 'File-Open' later. This is done by typing the file name followed by the name W32APBLD. For example, to run the command and open the project example enter dialogue:

W32APBLD \cetlib\samples\cetdemo.prj

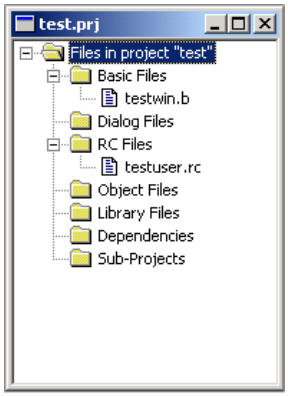
5.4 Create a project

The command 'File-New' serves to create a project. When you run this command will close any open windows containing source code, dialog or other projects. If changes have been made in those files the program will prompt you to save before closing.



A dialog box will appear called "New Project Settings' in which to write the name of the new project, which can be up to eight characters. In our example, we have given the name 'test'.
If you choose the button 'Advanced', another dialog box appears in which you can specify: the name of the executable file, by default the same name as the project name, typing another name not add extension since. Exe is added automatically.
You can also specify the name of the resource files that your project will use. By default, W32APBLD create a new file pair of own resources and this new project will be a

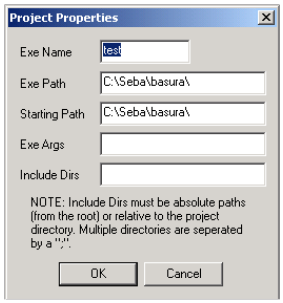
copy of the files cetwin.by cetuser.rc the \ cetbin (default), the name of the new project located in the directory local. If the name of the project and the executable are the same, need to change the names of the resource files in order to avoid a conflict between the source files containing BASIC code.
If you do not want to use a copy of cetlib files, you can:
1. - Write the name of other resource files, previously created or imported using the 'import' that shows a dialog to select files. This method is recommended when a file is modified, you want to use in this project.
2. - Choose "Create an Empty Project", (void project resource files) when you already have a set of resource files that will be added to the file using the 'Add'. This method is recommended when you create a project that will work with other executable files (subprojects).



Note that if you add the compiled versions of these files (extensions. Obj and. Res), W32APBLD not create additional copies in the current directory.
In our example, the names of the project and the executable file are the same, so we have introduced testuser testwin and filenames as resource. (The program automatically adds the. By. Rc), thus avoiding a conflict with the name of our source file test.b

When you press the 'Create', a dialog box will appear like 'save as' it will ask the name of the directory where you store the project file. Prj.

Once you have created the project, a window that displays the files in the project tree. Folders tree identifies a file type: Basic, dialogues, RC, objects, libraries, dependencies and subprojects.
Folders that are preceded by a plus sign indicates that the folder is expanded, listing all files that belong to such project. click on a minus sign, or double click the folder to expand it.
If you right-click on the folder 'Files in Project ..', then, will the global properties of the project. These properties are:



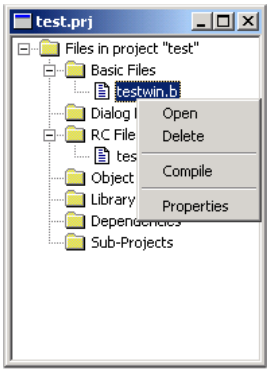
1. The name of the executable file.
2. Any argument that is passed to the executable.
3. # Include directories. If you are using include files that do not belong to the project, be sure to indicate your full path (absolute path name or relative to the project directory) in this section. Separate multiple paths with a semicolon sign.
4. The way of beginning. This will be the working directory of the application, when started running.
5. The path of the executable files. Store here any way different than the default path (project directory) executable files.
It is important to remember that these properties are global to all files of the project. If you use the same directory for all include files, find more useful to establish the absolute path in the section 'Global Directory Project Include' button to execute the command 'View-Setting'. Note that the change will not take place until the next time you run W32APBLD.
Each folder in the project tree has its own global properties (the type of files they represent). For example, the-c flag is always used by default in all files of type BASIC. If you add the flag-Zd to the global properties of this folder, then enable the debugger to all files. B application.
Each file has its individual properties. Any property is added to the global properties of the type of file during the build process.

5.5 Adding files to project

The F10 command, or menu-FILES TO ADD PROYECT displays a dialog box to select the path, name, and extension of the / the file / s to be added.
It is necessary to take into account when adding a file:

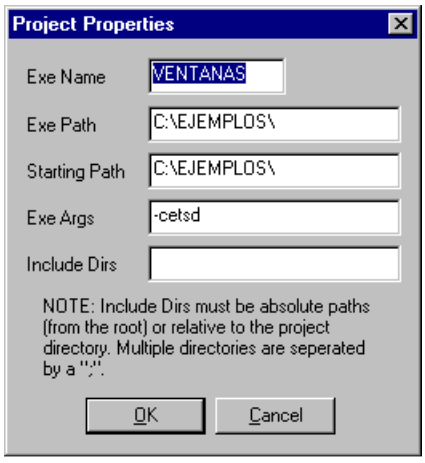
1. The dialogues and forms must be added by selecting the file extension. B, then the *. Idh be added automatically. If the file is added directly to a *. Idh an error is detected.
2. The source files are added by selecting files extension BASIC code. B. If these source files contain # include statements, these files are automatically added to the build process (build) as dependencies. If you create a new unit after the project has been compiled, you should use the command 'BUILD - UPDATE ALL DEPENDENCIES' to add to the project.
NEVER use the ADD option to add file extension. H,. Inc., and / or. Cpu, otherwise an error occurs. To avoid confusion with extension include files. B, you can rename these files with another extension to distinguish them from files. B BASIC code.

3. - You can add resource files, if you have checked the option 'create an Empty Project' (create an empty project). If you add these resource files (. By. Rc) When you compile the project (Build) then also automatically compiled and stored these files (the compiled) (. Obj and. Res) into a subdirectory. BLD. In this case they are added resource files, the environment does not back up files renowned system resources as would be usual.
4. - We recommend that you add the *. OBJ (already compiled) when you go to add a library created with the utility CETLINK32. This file can be modified during operation and relinkado compiling and linking (Build). If a *. Obj is part of a library, the dates of both files should be modified. In that case the environment relinka all files in the library.



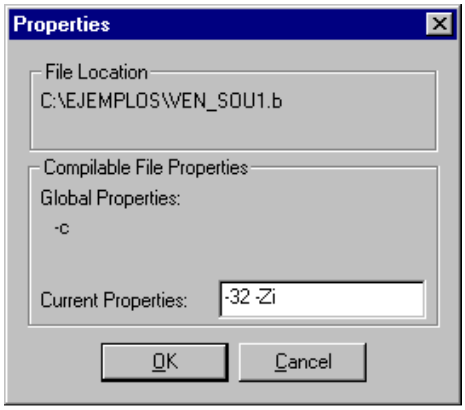
5. -The subprojects to be added as executables. These statements are called with type CHAIN from some source files (*. B).
You can add several files at a time: either all at once by selecting them with Shift + click or selecting some control + click.
When a project is open, you can create source files, edit them, and dialog windows. At the moment we are going to close or save the environment will prompt you to add it to the project.
This also occurs when you create a dependency file extension *. Inc. * Ho *. Cpu. In this case you should answer you do not want to add the file to the project because, as mentioned, would give an error message.
Finally, we strongly recommend that you NOT use the Save As option to rename files, this could lead to problems with projects using the same. We recommend that the project be deleted (delete), will be renamed from the command line (*. By *. Idh if dialogue), and finally back to the project reañadan. After adding a file to a project, when you press the right button options are 'fast' the type of file to which you pressed:
• Open Edit the file.
• Delete removes the project file.
• Compile Compile the file using the flags specified in the properties of that file.
Properties Specifies the file properties. They are different project properties properties files:

Project Properties:
Displayed by pressing the right mouse button over the project root + properties:



1. Exe Name: name of the executable file to generate.
2. Exe Path: path where to store the executable file.
3. Starting Path: working directory.
4. Exe Args: execution arguments.
5. Include Dirs: search directories include files, indicating the paths can be relative to the working directory.

Properties files of the project:
are obtained by pressing the right mouse button, and then clicking Properties.



1. File Location: monbre and file path.
2. Compilation Flags

5.6 Building and maintaining a project

After a project is created, the menu 'Build' can be used to perform a set of operations related to projects. These are:

Compile to File (compile a file). Compile the file using the default flags specified in the properties file. Note that to compile a dialog to select the file. B, not the file. HDI. The-c flag is used by default.
Remember there are two levels of properties: global and individual properties file. In fact, there are three levels considering the 'Global Directory Project Include' the way of a global include directory for all projects.
Global properties affect all project files. During compilation, the compilation flags for individual properties files are concatenated to the specified global properties.
For example, suppose you want to use the debugger in a file. To enable the debugger, add the flag-Zd the properties of an individual file. If you add the flag at the properties of BASIC type files, then the debugger will be enabled on every file extension. B. (This feature is explained in detail in the next chapter). The individual properties of a file can 'crush' to global properties. For example, if you specify the-c flag (create and linkar files. Obj), you could use the-s flag on the individual properties of the file, forcing the build environment to complete the process after creating the files assemblies. asm.
Build the Project (Building the project). Compiles all programs and generates executables. The directory <project name>. Bld is used to store the compiled files. Obj, resource (. Res) and the text file with the error messages of the compilation, ie w32error file.
During subsequent compilations, W32APBLD check all dependencies and recompile only those files that have changed since the last compilation. During this process we will try to delete the executable file. If the file can not be deleted because of an access denied, (which usually means that the file is in use at the time) compilation aborted and a message is displayed indicating the reason.
Note that W32APBLD can not detect changes to files that are referenced in resource files (cetwin.by cetuser.b). For example, if cetwin.b is modified to display a new button in the toolbar, or have changed the *. Bmp or *. Ico, W32APBLD will not notice, and you are, responsible for opening resource file and save them to W32APBLD recompile these files in the compilation process.
Force to Rebuild All recompile all project files, although these have not been modified.
All Dependences Update (Update all dependencies) when find a # include statement, look for files in the folder BASIC type files and updates certain dependencies. (Use this command to add new units after the project is built).
Execute the Project (the Project Run) Run the project. If win32app detects that a file has been modified since the last time you built the project (comparing dates), then displays a message asking if you want to upgrade the project.
Build the subproject Subproyects automatically opens and executes the command 'Build the Project'
Rebuild All Subproyects Automatically opens each subproject and run the command 'Rebuild the Project'.
Whenever a file is compiling or building a project, a dialog box appears to indicate that the operation is being performed. If this operation is successful, you will return to the editor window without displaying any message. However, if errors are detected, W32APBLD w32errorfile edit the file, to show the list of all those mistakes.
Note that the dialog has compiled project to minimize and cancel buttons. If you minimize the build, it will be in the background and when finished, the program will automatically reset. If you cancel the compilation, the program will end after the command is executed the current command. This option will not abort the command.

5.7 Create subprojects

Typically w/32 programs consisting of a set of individual programs are called via CHAIN, RUN. Although the projects are created to give a single executable, called programs can be added as subprojects to facilitate application maintenance.
For example, suppose that the program does a CHAIN mainprog.exe to subprog.exe. Then, you should follow the procedure below to add it as a subproject:
1. Create a project to mainprog.exe
2. Create a project to subprog.exe. Highlight "Create Empty Project" if the program will use the same menu (and toolbar icon etc) mainprog.exe. In this case, add the compiled resource files (the *. Obj and *. Res) that should be in your project directory called mainprog.bld. Otherwise, the project would need to have its own copy of the resource files cetwin.by cetuser.rc
3. Reopen the project you created for mainprog (mainprog.prj). Use the 'Add-Files' (add files) and selecting the file subprog.prj in dialog.
The menu 'Build' has a special command 'Build-Subproyects'. When used, W32APBLD will show a message that you are checking dependencies. If the file dates are subsequent to the executable, the subproject will be rebuilt. If the executable is later date, the program will continue checking another subproject, if any. Selecting the command 'Build - Rebuild All Sub-Proyects' all subprojects will be rebuilt, regardless they are current. In this case, no user interaction is required.

5.8 W32 examples Projects

W/32 Application Builder includes a number of example programs that have been created to illustrate many features of this product. There has been a special example program, a menu of programs other example. This program is called W32DEMOS example.
Projects have been created for each of the programs examples. All these projects files (including executables) are stored in the \ cetlib \ samples, by default.

Proyecto	Descripción
w32demos.prj	A special menu program that runs other programs such.
cetdemo.prj	Dialogue example described in chapter w/32 Dialogue Editor.
invoice.prj	An invoice program, with the display format example shown in Chapter w/32 Dialogue Editor.
calc.prj	A calculator program, which contains the event processing example of buttons displayed in the dialog editor chapter w/32.
odbc.prj	A program that uses the features described in Chapter ODBC.
suprint.prj	Un diálogo ejemplo cómo se puede utilizar la función cEnumPrinters para conseguir información aceca de las impresoras disponibles.

5.9 Rename, Copy or Move existing projects.

When you create a project, all project information such as your name and location are stored in a binary file extension *. Prj. When you add a file to a project, the file is updated. Prj to include the file name and location. Since individual source files can be located anywhere in the system, all locations within the file are stored as absolute path.
The information in this file will be invalid when the project is renowned, copy or move to another location (using operating system commands). Fortunately, W32APBLD can detect these changes that have occurred and update the project file according to changes.

5.9.1 Rename a project

Use the operating system command to rename the *. Prj. This file must not be open while being renamed.
Call W32APBLD to open the file with the new name. The name of the file to be opened will be compared with the name of the file is stored in the *. Prj. When the names are different, W32APBLD warning him and displays a message advising that the file names are updated. If, at that time, you select 'Cancel', W32APBLD not make any changes. Select 'OK' to W32APBLD change the names that are stored within the file *. Prj.
W32APBLD also checks the executable name. If it is the same as the original project, renamed to put the new project name.

5.9.2 Move or copy a project.

You can create a new project and add the files or rename one and follow the process described above to rename the project.
If you rename the project, W32APBLD detect that the way the project is different
and show a message that the stored internal roads have changed. If you select OK at the time, the program will update the roads, checking the paths of all files of the project. When you find a file with the old road will also have been moved, and will change the way of updating the new road project.
Note that there may be cases in which this procedure does not work well: could be files that were not moved along with the *. Prj. There may also be files that were not in the project directory (file sharing).
If there are only a few exceptions (eg, the project only has a shared file that is not in the project directory), then you can follow the procedure for moving the project, then delete the file sharing project, and add it again . This is assuming that all the old road, now, is not valid. If the road has not changed, obviously, you would not have to do anything.
If the project has been renamed and moved, W32APBLD develop both procedures at once.

Chapter 6 The debugger W/32

6.1 Introduction

W/32 Debugger is a special feature that is only available from within the integrated development environment w/32 (W32APBLD.exe). After creating a project, you can use the BASIC debugger to locate logical errors and execution.
This chapter will give you enough information to use the code editor and debugger possibilities. (This chapter assumes that you already know the features that are explained in the previous chapter).

6.2 Using the Source Code Editor

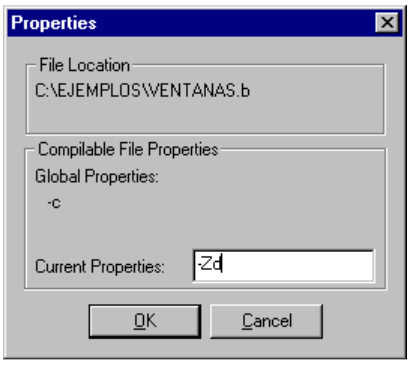
W32APBLD has a source code editor you can use to create, view and edit your source files. Menu command 'File' (file) has the following elements, which will simplify the process:
Command Function
If you select File-New 'source' (source) will load the code editor with a file called "sourcen" where n is the number of new files you have created and not yet saved.
File - Save Save a file. If you are saving a new file (sourcen), then enter the name you want to give the file. The extent. B always added by default.
File - Print Prints the text file ..
File - Find in files ... Allows you to search for text within a file. The 'Match Case' can be used to search note the difference between upper and lower case. You can also specify the directory to start the search and check in subdirectories.
The type of files to search (eg extension files. B, and. Inc.) can be selected from a list of file types yourself or by entering the extension of the files to locate. (Use a sign semicolon to separate each type. For example "*. Foo; *. Bar").
When the search begins, a dialog box appears with the result. This window specifies the found files where the search pattern with file name, line number and the line of text that matches the pattern (shown below). At any time, pressing up a line, will jump to a window with the edited file and above the line you selected.
c: \ pos \ source \ mainprog.b (234) This is the text line ...
Dialogue can be minimized if you want to continue working on other tasks while the search is being conducted. To cancel the search, click the System button 'close', or the cross X in the upper right. Clicking on the option will exit the dialog close and delete the search results.
Menu commands 'Edit' will facilitate changes in their files. The commands are:
Command Shortcut Key
Find (search) Ctrl + F
Find Next (search next) <F3>
Replace (replace) Ctrl + H
SelectAll (Select All) Ctrl + A
Go to Line (go to line) Ctrl + G
Note that the command has been added 'Select All' so you can select the full text of a file without having to move the mouse across the entire file.
In addition to the standard features of Windows type editors, W32APBLD editor allows the Insert key can be used to switch from insert to overwrite mode and vice versa.
By default, the characters stop down every 8 characters in the Windows Notepad program. To change this setting, use the 'View-Settings', and enter a value of "Num Chars" (number of characters) between 1 and 16 in the "Text Editor Tab Stops" (tab stops text editor) . (Note that this change will not take effect until the next time you run W32APBLD).

6.3 Using the debugger W/32

W/32 The debugger allows you to insert breakpoints in your BASIC source code, then consider it at runtime the value of the variables that have been used in that file.

6.3.1 Enable debugger.

To enable the debugger, use the flag-Za (to be case-sensitive). For the information of the debugger is inserted into the BASIC source file during compilation.



Debugging can be enabled for one or more source files. To enable the debugger for all BASIC files of your project, set the flag-Zd icon in the properties 'Basic Files'. If you only want to include debugging information in some files, enable the flag in the properties of each individual file. When a source file has debug flag enabled, the file icon will be red instead of blue, as would be normal.
Note that the debugger can be used only during the development phase of your project. From now execute the option can not be used. Therefore, when the program is ready for distribution, be sure to disable this feature and run the command 're-build' (and see) your application.
After compiling the files, press the command 'Debug' menu. The commands that are available are:
Point Break (stop) Inserts a breakpoint at the current cursor position.
All Clear (clear all) removes all breakpoints in the project. You can also remove them one by one from the list that displays Show All, press the Delete key
Show All (show all) displays a list of all breakpoints inserted in the project files.
Wath displays a window containing a list of all the variables that are being used in the file. This command is initially uninhabited.
AutoWatch dialog window variables (Watch) is automatically displayed whenever there is a breakpoint during execution. This is the default behavior.
Continue when the program is stopped upon reaching a breakpoint, this option causes the program to continue its execution.
Step mode executes the program step-by-step, making a stop at each line of code.
Start the program begins execution and debug operation.

Chapter 7 The dialog editor W/32

7.1 Introduction

The dialog editor w/32 is a special tool that allows you to create input formats (which we refer to as the display format) and dialog boxes so that their programs can have a graphical user interface like any other Windows application .
A display format is a graphical login screen (based on pixels) used to replace the main window, text-based, for an application. The format can be full screen or can be set to the desired size using the dialog editor. In any case, the window can be moved and resized. Any time you are active a display format, have full access to the menu bar or the buttons on the toolbar.
A dialog box is similar to a display format, except that the dialog is modal. This means that the dialogue box will be closed (eg by clicking OK or Cancel) to transfer control back to the program that called it, and the user can continue working with the rest of the application. Menu items, for example, can not be accessed when a dialogue is still open.
You can create a variety of dialogues. A simple dialog box is one that takes some text messages and waits for the operator to select a button. A dialogue box complex could have a large number of controls (input fields) along with images, leaving waiting for the user to make an entry using mouse or keyboard.
Basically, dialog boxes can be used at any time when you want information to or from the operator, without this affecting what is currently appearing on screen. In other operating system environments, could have opened a window based on text (using the windows of Phase One), taking some texts, performing an operation like LINPUT and then closing the window when you are ready to continue. Under Windows, this will be done with a dialogue box.
Both dialog boxes as display boxes consist of a group of controls. Each type of control has a set of properties that define its behavior. Some controls will draw on screen simply texts serving as a label to identify other control. Other controls are designed for operator information. Checking controls, Combo Boxes Listed are some of the special types of control that can be used to simplify the data entry process.
For each dialog box or display format that you believe, the dialog editor w/32 generate a BASIC subroutine that will handle the display and operation of dialogue. When necessary, this BASIC code may be modified to suit your special needs. Examples have been added to illustrate how programs can be performed.
The information in this chapter is organized into the following sections to help you learn how to use the editor and to develop their applications as quickly as possible.

- Use of general editing features and menu commands.
- Creating a dialogue or a display format.
- Defining the types of controls.
- Change the appearance and manner of implementation of controls.
- Using RunTime functions to modify the look and behavior.
- Modifying the code generated by the dialog editor.

7.2 Creating a user interface

When creating new applications or convert an existing program based on characters, the first step to create the user interface is familiar with the features offered by the dialog editor. We have included simple dialogs and display formats to illustrate some characteristics.
If you have already made text-based programs, the next step would be to use the Import Template Dialogs (Import dialog templates) to convert each of the screens designed in text-based application, in order to obtain the corresponding windows dialog and can be loaded into the dialog editor to modify their appearance, control properties etc..
At this point, I'd be prepared to modify the source code and create a display format model that includes all registration search routines and data validation and formatting generally used in their programs. These routines plus those that are specific to the program, such as input-output operations to files can be stored in external files that are defined with # include statements (in this way can be used comfortably from foals programs).
Probably, you will want to customize the framework to display another menu, toolbars, etc.. which automatically gives CET programs. A previous chapter of this manual explains how to make these changes.
Once you are satisfied with the operation of its program model, create (or convert) your next login dialog. At this point, you should use the Copy-Paste (Copy-Paste) to copy code quickly (or inserting # include

statements) of the model to new dialogue format. This process is simplified if you open several windows simultaneously with different codes in the editor. The code of your original source file can also copy and pasted into new files # include easily. (In that case, we recommend that you delete the line numbering)
The advantages of using this product are essentially two:
- Windows applications are performed with user interfaces that look and function the way that the user prefers.
- Applications are made highly structured, event-driven, easy to maintain.
Now, off you go ...

7.3 The dialog editor w/32

The w/32 dialog editor is invoked from the integrated development environment running the W32APBLD.exe. cetbin from the directory or by selecting the item from the menu group application builder w/32 CET (CET w/32 Application Builder).
Note for Windows versions 2000 or higher: because of problems with the product W32, W32APBLD not allow proper compilation when it is called from a command window. This problem does not occur under Windows 9X or NT
Note that although this chapter is not intended as a tutorial, we recommend that invokes the editor and go attempted to test the features as they are explained, especially if this is the first time you used a tool like this.
After running W32APBLD, the menus of typical commands such as the File, View and Help. Although there are a few buttons on toolbars that are initially disconnected.
Load the editor by selecting the File-New command and option dialog. At that time other commands appear in the new menu and toolbars.
Besides the main tool bar, the dialog editor has two special toolbars to streamline the design process: controls toolbar on the left of the window and the toolbar just above the bar Status (Status). All toolbars can be changeable or movable.
The menu display (View) has options to show or hide any of the toolbars and the status bar (status). If you changed the "visibility" or location of any of the toolbars will program the same way the next time you call the editor. Reset Defaults command (to reset the defaults) can be used to make visible the status bar and all toolbars and positioned in the default position.
The control commands and viewing will be explained in detail in the following sections.
For now, note that each of the new dialogues contain two controls, an OK button and a Cancel button in the upper left corner. To move, resize or copy one of the buttons, you have to select it first, positioning the cursor anywhere on the button and clicking the left mouse button.
The selected button control will now surrounded by a blue rectangle with buttons resize (solid blocks) around the perimeter. In the status bar (status) displays information about the selected control: Your ID number (which will be explained later), position (top and left coordinates) and size (height and width).

7.3.1 Moving a control

To move a control, first position the cursor inside the selected area. When the cursor changes to a four-way arrow, you can move the control to a new location in one of the following ways:
1. Clicking the left mouse button and dragging the control. A dotted rectangle appears to help you move or relocate.
2. Using the arrow keys to move the selected control in a specific direction. This is the recommended method to control more precisely the position of the controls.
When you're done, deselect the control clicked on an empty area in the window.

7.3.2 Resizing a Control

To resize a control, select it and move the cursor over one of the resizing buttons. When a two-way arrow, click and drag to resize. You can also resize a control by pressing CTRL and an arrow key. In any case, tap outside the control to complete the operation.
The command-Grid Layout Size (mesh size) is used to change the increment to be used when a control is resized with the arrow keys. The default size is 5 pixels. The resize a control with the mouse is always limited to + / - 5 pixels regardless of the selected scale of the time.
While moving or resizing controls (by dragging with a mouse or using an arrow key), the size and position of the controls will be continually updated on the status bar (status).

7.3.3 Copying a Control

To copy a selected control, first use the Edit-Copy command menu. Then select Edit-Paste command and move to the position where you will place the copy. Because all controls have a set of properties, control copying its properties are copied except ID values and BASIC variable names.

Later, you will find that this is a handy feature when you design a form that contains a set of related controls such as address1, address2, city, etc..

7.3.4 Deleting a Control

A selected control can be removed using the Edit-Cut command from the menu, the button on the main toolbar or the Delete key. (Also available is a delete command in the menu controls (Controls)).

7.3.5 Selecting multiple controls

You can also move, copy or delete an entire group of controls. You can select a group of controls for any of the following methods:
1. Select a simple control. Then, select the CTRL key and click on other controls. If the new control is not part of this selection is added to it (and enclosed in a blue box). If the control is already part of a group, will be deselected.
2. Use erasing technique dragging around the contour of the desired controls. A dotted rectangle will appear on the trail of the mouse. Release the mouse button when all controls have been fully enclosed by the rectangle.
Any control that remains within the defined area will be surrounded by a blue rectangle. If you get the controls you want, leave the selection process, clicking anywhere outside of the selected controls. From this point again begin the process of entrainment.
Note that when a group of controls that are selected, "Multiple" will appear as the ID of the status bar (status). The position and size fields indicate the bounding rectangle, the smallest rectangle that encloses all controls.

7.3.6 Undo and Redo options

The dialog editor w/32 remember all editing changes have been made. To fix a possible error select Edit-Undo command (undo) or click the Undo button on the Status bar (state). If you decide you would like to keep the change simply click the button again to make "Redo".
Any operation can go backwards, and any operation that may have gone back to again. When using this feature later be careful because some operations such as changing properties of controls can not be visually checked, but can be remade or can go backwards.
The number of operations and unpacking are limited only by the size of virtual memory and can grow to a very large amount. Saving the file will erase all buffers and unpacking. (We recommend that you periodically save the file).

7.4 Opening an existing dialog

File-Open command displays a window with the different types of files, select the type you want to open. There are two files for each dialog you think. These files are:
1. File extension. Idh, containing the code that is used to create and display the dialog. Selecting a file extension. Idh be charged the dialog selected within the dialog editor.
2. Associated file extension. B, which contain modifiable source code. Selecting a file extension. B, the code will be loaded into the text editor.
You can have both files open in different windows at the same time. This feature will save you time multiwindow development.

7.5 Save a dialog

Each box has a property called prefix (prefix). By default, this value is used as the name of the file dialogs. (This property is explained in the section Setting dialog or display formats)
When you try to save a new dialogue, the editor will display a dialog box like 'Save As'. This feature allows you to check that no other file with the same name. If there is, cancel the operation and change the prefix of dialogue.

7.6 Utility: Dialog templates.

The build environment window includes this option will help you build templates dialogues from images of your old data entry screens, this utility will speed much change old programs windows textual programs with multiple windows interface:
This mechanism makes all text fields are converted to type controls label followed, optionally, by a control of type text editor. From here you can access and modify those controls.
Create templates dialogues is a simple process 2 or 3 steps:
1. - First, create an ASCII text file that contains the image to be converted. This is done using a special-cetsd argument to run the program compiled with version 8.14. For example:

client-cetsd
When you leave the image you want to capture, press ALT + PAGEDOWN (at once) and then the image is stored in a file called cetsd001.txt. If they were more of a screen that were caught then be stored in files cetsd002.txt, cetsd003.txt, etc. (These files are crushed in a new execution).
The text files are stored in the current working directory. If the application changes the working directory during execution, then you have to look for files.
Note that any graphic character that were used to create boxes on the screen will not display graphics as characters in the text editor. Add a control group if you want to create the same effect in the new window that you design.
2. - Edit the image file. This step is optional but is used to make modifications. For example, if you want to change the labels. If you plan to use keys in the program:
before: after:
1. Company & Company

Importantly default W32APBLD uses three spaces as field separator text, therefore, in the following example consider three items and give rise to three label controls ("1. Company", "[", "□ "):
1. Company [□
Although it is very easy to delete extra controls in a dialog, we suggest you use the property Pairs Conversion (Becoming then) to convert extra characters such as brackets to blanks before the file is converted. We recommend it especially if you plan to create an edit control text for each item.
3. - Run W32APBLD and select the Import Dialog Template from the File menu. You can click the Browser button to find the file. Txt easier. The default extension is. Txt. The file will be converted and charged with default properties dialog. All items will be converted to text labels controls.
The Settings command in the View menu can be used to modify the conversion process to use the next time you run the Import Dialog Template. The properties of the "Dialog from Text Template" are:

PROPIEDAD	*EFECTO*
Conversion pairs	This property can be used to convert one character to another. S Use the comma to separate elements decimal conversion pair, eg FROM, TO and use a semicolon to separate multiple pairs. Example: 93.32, 91.30.
String separator	The default is to use three blanks as separator two text elements. A blank character is represented by \ b. For example if you specify in this property ": \ b \ b \ b" will be assumed that each text element will end with a colon followed by three spaces.
Edit Control	This property makes adding a text editor after each text control (label).
Fixed font	This property will make a fixed font is used in all text controls.

When you turn your screen into a dialog, you can change all the properties of the controls as well as normal dialogue.

7.7 Defining a dialogue or a display format

W/32 dialog editor lets you create display formats and dialogue boxes. A display format is a graphical input (based in pixels) that the program will use to replace the text-based screen of the application. Taking a graphical interface allows you to copy data from a Windows application like Microsoft Excel and paste it into one of their formats through the Clipboard.
Because a display format replaces the home screen of the Windows application, only one can be active simultaneously. Depending on whether the property has been opposed maximization, the format may either be full screen, or have the format as it was defined in the dialog editor. In any case, the window can be moved and resized.
Any time display format is active, will have full access to any of the commands in the menu bar and the buttons on the toolbar. A format can be minimized or maximized.

A dialog box is similar to a display format, except that the dialog is modal. This means it has to be closed (eg punctured OK or Cancel) before control is transferred back to the program that called it, so that the user can continue working with the rest of the application. The menu items, for example, can not be accessed while the dialog is still open.
You can create a wide variety of dialogues. Basically, dialog boxes can be used at any time when you want information to or from the operator, without this affecting what is currently appearing on screen or in other active dialog boxes.
Each time you use the File-New command (File New), the publisher will assume you are creating a new dialogue and assign a default set of properties. To get to the screen I change a property, double-click the left mouse button in an empty area of the window. (Performing a double click on a control, will show the properties. Defining controls will be explained in the next section).
All dialogs have the following properties:

7.7.1 Tittle

The default text "Dialog Title" (title of the dialog) can be changed to describe the content or purpose of the dialogue. CDlgTitle function can be used to define or change this title at the time of execution.
If you define a format, use the title "Form View" (display format) while in the dialog editor. Upon execution, this property will be ignored and the format appears without title. To display a title during the initialization process, it will take cAppTitle calling from within the main application program, or from the subroutine contained within cetwin.b cPreInit.

7.7.2 Prefix

W32APBLD assigns the default prefix "dlg" and a value indicating the number of dialogues that have been created. For example, the prefix for the first dialogue will "DLG1".
When you try to modify the code, it is important to remember that the prefix must be unique, as these characters will be added at the beginning of all the names of the variables $ GLOBAL and subroutines used in the program dialog. If you change the prefix in a later session, then a message appears advising you to update the dialog using the 'File-Rewrite. HDI and. B' (rewrite files. HDI. B)
The prefix name will also be used as default files. HDI. B generated when the save dialogue. For example, the code for one of the examples dialogues is "dlgCET". The name of the file on disk is dlgCET.b.

7.7.3 Type

The default option is to create a dialogue that is called from the main program or from another box. The dialog box will always remain focused on the Windows application.
If you specify a display format, the format will be used to replace the window of the text-based application. The maximization property will use the full screen format. Otherwise, the application window will format, starting exiting the screen upper left corner. In any case, the user can move or resize the format. (The dialog boxes are only those that can move).
Note that has a variety of functions w/32 windows, if you want to modify the program at the time of execution, for example are functions maximization, minimization, replacement of the screen at runtime, etc. See the chapter on Windows w/32 Functions for details.

7.7.4 Size

Besides forcing the main application window to fill the entire screen (by setting the maximization), you can set, explicitly, the width and height.
Note that the default feature size comes with 300 x 200 pixels. As you resize the dialog box, these sizes will change each other. (Refer to implementation notes in the appendix to get an idea of how to think in terms of pixels, and for an explanation of the command argument (-wsize) that can be used to change this default at the time of execution).

7.7.5 Source file

The files in the dialog are explained in detail later in this chapter. For now, just note that there are two additional properties that can be set, and that will change the BASIC source file to be generated. The two properties refer to these two options:
1. Not to ask for interactive control values (Interactive controls are those that capture user input).
By default, generated CALL statements to automatically search the values of all interactive controls before the end of the dialogue. This means, you do not have to modify the source to get the information passed to the main program with $ GLOBAL statements.

The control dialogs this default feature. However, display formats, typically have to modify the source programs by hand to recover the values of interactive controls, according to their specific needs. In this case, update this to the editor does not generate this code also.
2. Not generating the control variables $ GLOBAL:
By default BASIC control each variable is defined as a global variable (with $ GLOBAL statement). This same statements $ GLOBAL also be written in the main program so that the information can be accessed from the dialogue. (Edit-Copy command can be used to copy the sentences $ GLOBAL clipboard. Then be pasted into the main program using Edit-Paste).
If this option is used, no statements will be generated. In that case you should manually enter these statements to all global variables that are used in the main program or in other dialogues in both the main file and in the file dialogs. This procedure is explained in the section examples of source files Dialogues.

7.8 Defining controls

All dialogs and display formats consist of a set of one or more controls, some controls can be simply labels.
The menu controls (Control) and the corresponding toolbar contain commands that are used to introduce specific controls. Then there are the following commands:

Command Function
Test (check) to check the design and operation of the dialogue without having to compile the program. The dialog will automatically take three dimensional appearance that is common to all Windows applications.
Select (Select) to exit the selection process. For example, if you selected "combo box" from the menu by mistake, immediately go back to the menu and choose "Select" to exit this operation.
Delete (delete) Deletes the selected control dialogue. The delete key (Del) can also be used to remove a control.
The rest of the menu items can be used to introduce specific controls. The controls available can be divided into two types: static and interactive. A static control does not allow operator input. (Text and controls are static group). An interactive control by the user accepts an entry or selection.
All controls have a set of properties that affect its appearance and the way it works. These properties can be set at design time, dialogue editing and clicking with the mouse to control properties, or, at runtime, using some functions described in section Files dialog sources.
To avoid redundancy, we define here the two properties common to all controls:
1. Ownership: variable BASIC, is a name like EDIT100 $ and $ LBOX300 indicating the type of control and the ID number that define it. These names can be changed to make more sense for you as a programmer.
Each variable BASIC statement will result in a $ GLOBAL automatically. The $ GLOBAL specify the prefix of dialogue (dialogue prefix property) followed by the variable name BASIC. This statement can be used to pass information between the dialogue and the main application program.
Note that the prefix of the dialogue (eg DLG1) is added at the beginning of each global variable that the variable to be unique to that specific dialogue. This prefix will be used in all the names in the source BASIC subroutine.
2. The unique ID value should not be changed because it is used to identify the controls in the dialog program. These numbers will be within a specific range depending on the type of control. Example:

Tipo de control	**Valor del Identificador**
Radio buttonsGroups	1 to 99
Edit text	100 to 199
Command buttons	200 to 299 (200 = OK y 201 = Cancel)
Lists boxes	300 to 399
Scroll bars	400 to 499
Labels	500 to 599
Frames	600 to 699
Combo boxes	700 to 799
Check boxes	800 to 899

Radio buttons	900 to 999

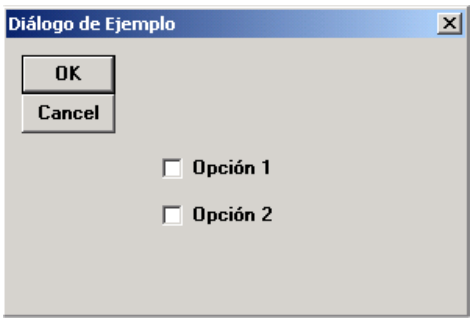
All dialogs and display formats consist of a set of one or These values produce no control ID conflict with the values associated with menu commands (ID_WINSUBn). ID Control values are processed separately when in the specific section for client code generated by the BASIC program editor dialog. (See section Source File Dialog).
All controls also have a set of ten user-defined properties that appear when you click the right mouse button. These properties can be used to enter information specific to the particular control. There is an option that lets you change the labels for these fields. example:

Campo	Etiqueta	Ejemplo
1	File source	cp\data\custname
2	Font field	10
3	Name	costo
4	Format	N
5	Upper lower case	
6	Required entrance	Y (si)
7	Mask	####.##
8	Ask message	introduzca la cantidad pedida

The dialog editor loads the information defined in these properties in a matrix which is accessible when you need to draw on the screen, validate or format a data in a specific control.
The procedure to use controls and their properties will be explained in detail in the Source File section of the dialog.
Adding controls to a dialog is a simple three-step procedure. simply:
• Choose a menu control.
• Move the cursor to the desired position and click to select the control.
• Double Click on the form control to enter their property. The user-defined properties displayed by clicking the right mouse button.
A dialog box called dlgCET example is stored in the default directory \ cetlib \ samples. Because this dialog uses very different controls, is used in one of the next sections to illustrate the different features that are available. (To see the dialog, run CETDEMO or use the menu command-Test Controls after loading the file into the editor dlgcet.idh).

7.8.1 Check Box

The check boxes are typically used to allow the user to select from a group of options either all, some, or none can see the status of the box for display. The check box control can have two states: checked and unchecked, which is the default.
When the program is run, the user can select a check box by clicking anywhere within the control (in the box or in the title). Once elected, the state of the control varies. If it was not chosen will be and vice versa.
The active test box being used appear with a dotted line around the title. (The control is being used which has been more recently with the TAB key or has been stuck with the mouse. This feature will be explained in the Source File Dialog). The check box controls were used in the dialogue shown here dlgCET.b:



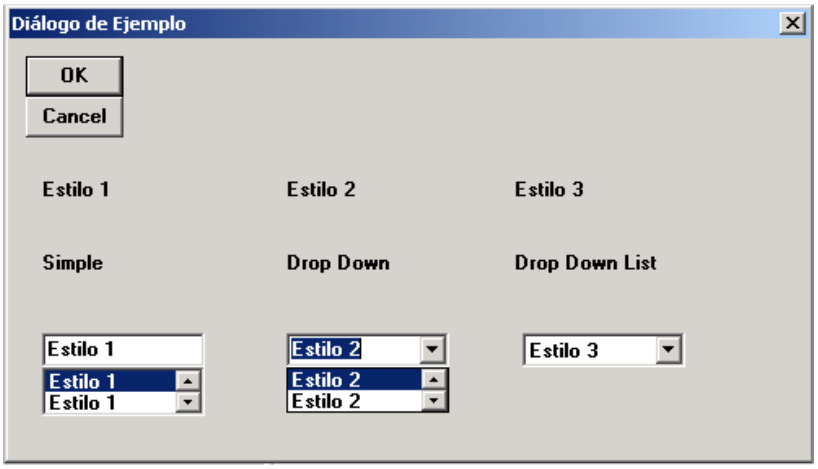
To add a check box, click the control menu and you will see a box with an arrow pointing to the upper left corner. This is a special cursor called "drop" that can be moved to any position of the dialog window. The arrow is used to indicate the top left position of the new control.
Click the left mouse button to leave the control positioning. The word "New" (New) with a small box will appear centered on the left edge of the control. Perform a double click (with the left mouse button) on the selected item to draw on screen and / or modify control properties check box.

The following information has been introduced to check the box "Disable" in the dialogue dlgCET.b:

Property Value Effect
Label Disable (offline) The text appears beside the check box to identify its content or use.
BASIC Variable $ DISABLE The variable used in a BASIC program to reference the data (after the variable has been defined with $ GLOBAL).
ID Control ID 800 The value associated with the control.
Initial State control is initially off without checking, which is the default.
A check box control is an interactive control. Because the information obtained operator has associated properties, such as BASIC and variable control number ID that can be used later to obtain the data.
BASIC variable to "Disable" (offline) check box has been changed to DISABLE CKBX100 $ $ so that is more descriptive and easier to remember. DISABLE means dlgCETDISABLE $ $ in the BASIC source code as the prefix of the dialogue is added to the name to make it unique for that specific dialogue.
The initial state can be set to control properties, in the example we set the initial state of the check box called "Option 1" to "checked". The other check boxes were "not tested" by default. The source files chapter explains the program dialog cetdemo.b example that checks / cleans the two boxes on the right if you check / clean the box "disable .."

7.8.2 Combobox

Combo boxes provide a list of options (as listed boxes), but also allow the user to enter an additional element. You can define an "initial choice" as a property of the control.
For example, the user may want to specify the shipping method, although as usual may appear by default, the user could choose from a list of other common forms or even introduce new.
There are three styles of combo boxes depending on whether or not the user can enter a value that does not exist in the list. The following dialogue illustrates each type:



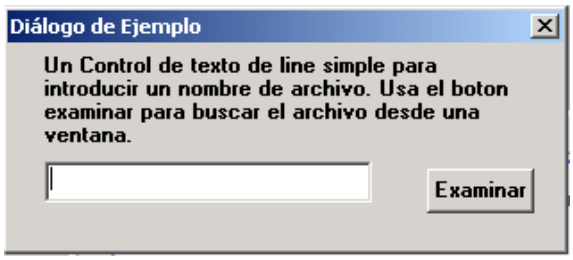
Use a) a simple box b) shutter (with scroll bar) for the user to make an entry as they have an edit control associated with them. Or, c) use a shutter control list that is associated static text.
A simple combo box control is always visible list box so that the control could be resized to show the number of options that you may want to appear.
The styles of blinds have a property that allows them to specify the number of lines you want to appear. The default is to display three lines once the user has clicked on the down arrow. In any case, the number of lines is a matter of design as can always use the scroll to view the rest of the element.
The alphanumeric display that appear on the list may be specified as a property of the control or change it on the fly at run time using function cAddComboContents. In any case, a call to the function will return the entry cGetCombo user-entered.
Note that the initial alphanumeric are limited to a total of 175 characters in length. If the total number of characters exceeds this length will then initialize the contents of the combo box at the time of execution. The number of elements can contain a combo box elements is 15000.
The example program "invoice" at the end of this chapter illustrates how the combo boxes can be used in an application.
Summarizing, the properties of a box may be:

Property Value Effect
Initial Alphanumeric Strings The alphanumeric display will be removed by the list.
The Initial Choice Default value is initially in the combo box editor.
BASIC Variable $ CMBO700 The variable used in a BASIC program to reference the data (after the variable has been defined with $ GLOBAL).
ID Control ID 700 The value associated with the control.
Fixed Font off the entry is displayed in a proportional font.

Simple Estyles control operate as a simple combo box.
Number lines 3 (number of lines). This has no effect on a simple combo box.

7.8.3 Command buttons

The button is one of the most used in the controls. It gives the user an intuitive display that enables easy operation - the user sees the label of the button and simply click on it with your mouse. Because the command buttons are so intuitive, normally used as menus.
The program will automatically detect which buttons have been punctured so that the desired option can be executed. (The program examples W32DEMOS menu offers the possibility to run example programs that use this feature)
By default any new dialogue has two buttons, one that is OK and the other is Cancel. The OK button properties are:
Property Value Effect
Label OK Button (Button label). Shows the label on the button surface.
ID Control ID 200 value associated with the control.
Default Button on (default button). Button that <Enter>.
End Dialog on (End dialog). The end button selection dialogue.
Note that the OK button has the property "default button" on. The dialogues in Windows normally have a default button is selected each time you press <Enter>. There can be only one "Default Button" in each box.
Both the OK and Cancel buttons have the property "End of Dialogue" set, so that the dialog will terminate and return values of any interactive control their respective variables when either button is selected. (The ESC key acts as the command 'close' in the Windows system, so it will also close the dialog).


Diálogo de Ejemplo
Un Control de texto de line simple para introducir un nombre de archivo. Usa el boton examinar para buscar el archivo desde una ventana.
Examinar

You can add other buttons to their dialogues. In dlgCET example, a button labeled "Browse" was added so that the operator can enter a file name (in the edit control specified) by selecting an item from a File Browser window Open (open file).
Note that the first character in the label (Browse) is underlined so that the button can be selected with a shortcut key. The label was introduced as "& Browse".
Each time a button is selected, the dialog program detects the event and calls the subroutine "BtnClick" with the global variable CMDID% with the ID of the button. This allows you to add specific code to check the ID and conditionally run the appropriate operations. (This process is explained in detail in the source files of the dialogue).

7.8.4 ContEdit text

An edit control is used to read an input by the user. Recognizes all as usual editing keys such as Delete, Insert, Backspace, Beginning and End If another line is introduced, automatically performs the bleeding of words and move the words that do not fit down the next line.
To add an edit control, click the menu item corresponding controls and release the mouse button at the position you want to place the control. When you press the left mouse button, the word 'value' is displayed above the control. A double `click' on the selected item shows the properties dialog editor.
All controls have the following properties in common:
Property Value Effect
Initial String "data" (Initial Alphanumeric). The data that appears by default when the dialog is drawn on the screen. If not specified, the default, the control is initially empty.
Max Chars "number" (characters max). Limit of characters that can be entered.
BASIC Variable $ EDIT102 alphanumeric variable that can be used in a BASIC program to reference the data (after the variable has been defined with $ GLOBAL).
Control ID 102 ID value associated with the control.
In addition to the properties "standard" of interactive controls that was explained in the beginning of this section, you can define a maximum number of characters that can be entered into an edit control. When the default value "0" is used, the maximum number is determined by the width of the control. If the property

hscroll Auto is active, then you can enter an unlimited number of characters. (There is probably a number, but this is very high).
An edit control can be either single or multiple lines. The default line is simple. Within dlgCET.b, has created a line edit control simple so that the operator can enter a file name. (See the box shown in the previous section).
The following properties are enabled only for single-line controls:
Property Value Effect
Auto hscroll on input automatically produce a horizontal scroll if the width of the control is exceeded.
An asterisk Password on every time you type a character.
Read Only on (Read Only). The content of the field can not be modified.
Fixed Font on (Font fixed). The characters on the screen in a fixed font instead of a proportional font (true - type) so that the user input from the screen can be pulled by a columnar format.
You can also create multi-line controls. If the control is sized large enough to make the screen all data have been entered, you will see a scroll bar by default.
The following properties are enabled only in multiline editors:
Property Value Effect
Want Return on (Necessary Return). It will require introducing <Enter> when introducing a data.
VScroll on see a vertical scroll bar.
Hscroll on see a horizontal scroll bar.
Auto VScroll on input automatically produce a vertical scroll if the width of the control is exceeded.
Test on Left (Left text). The text will be left justified within the control. The default option.
Center on Text (Text Center). The text will be centered within the control.
Right on Text (Text Right). The text will be right justified within the control.
Note that a single control line can only be aligned to the left.
By default, multi-control bleeding occur in the next line if the input exceeds the width of the control. If property is used VScroll, scroll bar appears when the control's size is not large enough to represent the data. Activating or HScroll VScroll properties are automatically activated and the corresponding Auto VScroll HScroll.
Note that each time you press the button in a dialog <Enter>, "default button" is selected automatically. Only a multiline edit control with the property "Return be" activated allow usable <Enter> to data entry. This feature also allows you to describe the field (using the newline character "\ n") when you need to load the input into an array.

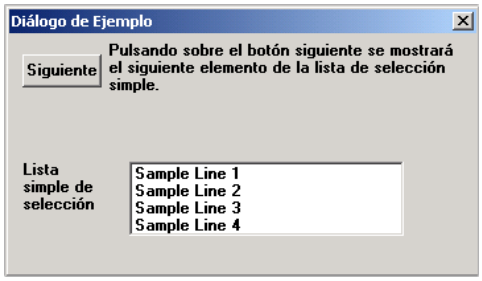
Note that each time there is any change in an edit control, a special event is generated. This means you can add code to the source file dialog to verify the data character by character as it is entered or wait for the user to select another control. This feature is covered in the section of the Dialogue Resource Files.

7.8.5 Frames

A control group can be used to alter the appearance of a dialogue by drawing a box to visually separate groups of related controls. Introducing controls in a control group (or defining a control group or covering around other controls) did not alter in any way the behavior of the controls.
To add a control group, select the appropriate menu item and position the cursor so that it is in the upper left corner. Click and resize the control until the desired size.
The group controls have only two properties: a title and a control ID number. Double Click on a title to change the default "Group". Enter a title or label is optional.
In dialogue dlgCET example, has created a control group labeled "Disable / Enable" (Off / On) around the checkboxes. It has created a control group without tag around text controls, editing and button controls that ask the operator to enter a filename.

7.8.6 Listboxes

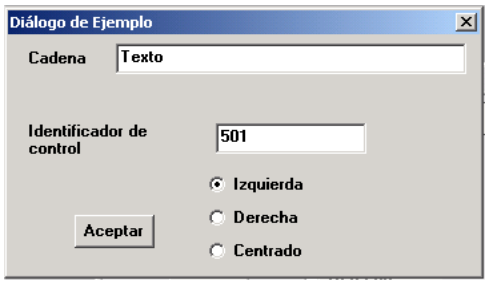
A list box is similar to a combo box, except that the user can not enter a reciprocating element. The user can select either nothing or one or a group of elements of the list.
Dialogue dlgcet example uses two list boxes, one for easy selection and another to make a multiple selection of items. The selection list box shown below simple with the Next button can be used to draw on screen the next set of options.


Diálogo de Ejemplo
Siguiente
Pulsando sobre el botón siguiente se mostrará el siguiente elemento de la lista de selección simple.
Lista simple de selección
Sample Line 1
Sample Line 2
Sample Line 3
Sample Line 4

The list box controls accept an array of alphanumeric (a virtually unlimited number of items), the screen takes the elements in a list of up to 15000 elements and allows the user to select. By default, the list box will have a vertical scroll bar.
The properties of the sample list box may be established as follows:
Property Value Effect
LBOX300% BASIC Variable The variable that can be used in a BASIC program to reference the data (after the variable has been defined with $ GLOBAL).
Control ID ID 300 The value associated with the control.
Sorted on (Sorted) Sorts the items before they appear on screen.
Multiple Selection on (multiple choice). Allows the user to select more than one item.
Fixed Font on (Source fixed). Take for characters in a font display fixed such that the list elements may be listed in columns.
No Integral Height Resize the box off list at the time of execution so that only complete lines are visible.
The special property of alphanumeric initial values be saved with no dialogue, only used for design purposes. We recommend that you use this feature to make sure that the size of the box is the right list and any label text control in the top of the list box is aligned with the columns in alphanumeric.
There is also a property of type fixed-width font to make it easier for screen representation of the elements of the list of columns.
The nonintegral height option allows you to specify the exact height of the list box may appear even partial lines if you have not properly sized. The default is a full height, if necessary, you can resize the list box at runtime so that only complete lines are visible.
By default, a list box lets you select a single element. The property can be specified multiple selections if you want to allow the user to select multiple items by entering a CTRL-click the LeftMouse. You can select a set of elements by clicking on the first item and then clicking on the last with a SHIFT-LeftMouse.
A list box can also have the property "Sorted" (classified). When specified, the items will be sorted before they appear on screen. When using this feature, it is important to remember that the index values of the selected item will probably not match the array index and initial. Although you can use the function to return the alphanumeric cGetlBoxString today it has been selected, we suggest that it is easier to use the BASIC MAT SORT to sort alphanumeric matrix before initializing the contents of the list box. This procedure will simplify your code because both index values are always the same.
When you add a list box control remember that your content is not set in the dialog editor!. This control must be initialized in the BASIC source code. In the dialogue example, this is done with a subroutine call dlgCETINIT cAddlBoxContents. The current alphanumeric must be removed by the screen are in cetdemo.b main program file.
Dialogue example also illustrates how the current contents of the list box can be changed at run time (by clicking the special button "Next" designed for this purpose). Special functions are also available to add, delete and insert an item into the list at the time of execution.
All these features are explained in detail in the section of the source file dialog.

7.8.7 Radio buttons

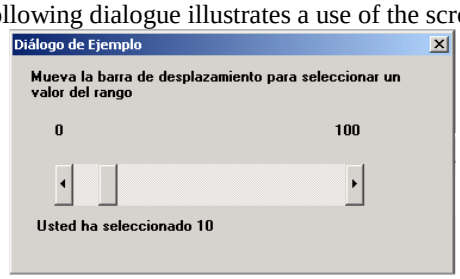
A radio button looks like a check box with a circle. While the check boxes are separate controls that may be enabled or disabled independently radio buttons are defined as a group so that only one may be selected simultaneously. If a radio button in a group is selected, any button that was previously selected deseleccionaría.
Radio buttons are normally referred to as controls option because options are often used as mutually exclusive. In the dialog editor, for example, radio buttons are used to specify the alignment property of a text control because only one of the three options can be set at the same time.



A dialog can contain any number of groups of radio buttons. Each radio button must belong to a group (and only one group). When many buttons are grouped, is assigned a radio group ID. Each button in the group will have that number group ID as one of its properties. (A group with zero value means that the button is not a member of a group).
The group ID is used to find the ID of the button that is being tested in a group (using the cGetRadioGroup). The return value will be zero if no button is being tested because they established an initial screen and the user did not select a.
Note that a radio button must be checked before it can get the focus.
To create a group of radio, first ensure that all the buttons are in consecutive order tab started from the top left button in the group. Then simply drag the mouse to select all button controls you want in the group, and select the command "Make Radio Group" (Creating a group of radio) in the Layout menu (PICTURE menu).
We suggest that you check the operation of each group of radio buttons with the test command in the Controls menu. If the buttons do not behave as you expect:
1. Select the "Undo" (undo all radio groups).
2. Check the tab order.
3. Remake all groups.
The properties for a radio control button are:
Property Value Effect
Label Text (Label). The text appears to the right of the button.
RDIO900% BASIC Variable The variable that can be used in a BASIC program to reference the data (after the variable has been defined with $ GLOBAL).
ID Control ID 900 The value associated with the control.
Group ID 1 (Group ID). The control belong to the first group of radio.
Initial State off (state). The button is not checked initially, which is the default.
The initial state of a button can be set as one of its properties or routine Init function using dialogue or cSetCtrlValue cSetCheck. To set or change the status of check in implementation, need to get the ID of the currently selected button in the group, using cGetRadioGroup function and setting the control to "unproven".

7.8.8 Scrollbars

The scroll bar controls vertical and horizontal are typically used to provide the operator a quick and easy way to select a value "approximate" a predefined range. The selected value is determined by the position of the scroll bar, which is a relative value based on the minimum and maximum properties.
Ownership of an initial value can be specified as the starting position. A scroll bar can be moved by clicking on any of the arrows. This changes the value +1 or -1 depending on which arrow is clicked. (If the scroll bar is very small, only arrows appear).
If control of the scroll bar is large enough, can also be moved using the "thumb" or the rectangle between the arrows. When you move this rectangle, the new value of the scroll bar is on the position of the rectangle. For example, if the box is in the middle of the scroll bar, the new value will be half the minimum range calculated by subtracting the maximum value.
If the user clicks between rectangle and arrow (referred to as a "page"), then one tenth of the range will be added or subtracted, by default. You can change the default page using cSetScrollPage function.
The following dialogue illustrates a use of the scroll bar.



The properties of a scroll bar can be:

Property Value Effect

BASIC SCRL400% Value The value that can be used in a BASIC program to reference the data (after the variable has been defined with $ GLOBAL).
Minimum Value 0 is introduced minimum value is 0.
100 Maximum Value The maximum value entered will be 100.
Initial value 50 The rectangle shown in the middle of the bar.
Control ID ID 400 The value associated with the control.
Horizontal bar appears on the screen horizontally, by default.
Vertical off This option has no effect on the appearance on screen when the smart button is ON.
As the moving scroll bar, an event routine is called to notify the new position. CMDID% value will be the ID of the scroll bar has changed and CET_EVPARM1% variable will contain the new position. CGetScrollPos function can also be used to obtain the position of the scroll bar. When necessary, you can change the range (minimum and maximum values) and the position at the time of execution and running cSetScrollPos cSetScrollRange calls. (These functions are explained in the section of the Dialogue source files).

7.8.9 Labels

A text control is used to display a pull by up to 255 alphanumeric characters. Although a text control can be used to head describing a group of controls, typically used as a label for the corresponding edit control. For this reason, it can reference a label control as is done in other development tools.
Text controls may be used to associate a memory pegs interactive controls having no associated labels. This is done by giving the text control mnemonic value (character by preceding the text with a amspersand) Then, when the user enters the mnemonic key, the focus moves to the next interactive control (in the tab order) (never the focus is in a text control).
To add a text control, choose the menu item, move the cursor to the desired location and click to leave control in that position. The word "text" will appear in a box with the default size.
Double-click the selected item to display the Properties box text control that lets you specify:
Property Value Effect
Text Alphanumeric characters that appear in the dialog box.
ID Control ID 500 The value associated with the control.
Fixed Font off the text that appears in a proportional font.
Left on Text The text will be left justified within the control. The default option.
Text Center will be focused on the text within the control.
Right on Text The text will be right justified within the control.
You can specify the alignment of text within the control selecting either left, center or right on the control's properties dialog text. The default is right justified. Only you can select a radio button in a group simultaneously.
The controls can be simple text or multiple lines. By default, the control will only have space for one line of eight characters long. When you enter a larger alphanumeric, control must have been long before the data is displayed.
Within dlgCET, all controls are multiline text. To carry out a multi-line text, simply select a control and extend it. The words automatically indented to the next line (if available) when the text exceeds the length of the control. To force a line break, enter a two-character sequence "\ n" when specifying the alphanumeric control properties dialog text. Enter "\ t" to tab if you want to create a column effect.

7.9 Changing the look and how to operate the dialog

The appearance of the menu and its associated toolbar contains a variety of commands that can be used to correct defects in the appearance and operation of the format. The commands are organized by category in the menu. Are covered here in the same order.

7.9.1 Alignment

The controls align one at a time adjusting is normally a difficult task and takes a long time. The first group of commands can be used to precisely align a group of controls selected according to the following rules. If only a control has been selected, these commands are grayed out on the menu.

command Effect	
Left alignment	aligns a group of controls arranged vertically so that it will be flush with the leftmost.

Align Right	Aligns a group of controls vertically arranging them so that they will be flush with the far right.
Upward	alignment horizontally align the controls under which it is more upwards.
Down	Alignment will align controls horizontally according to who is further down.

Depending on how they are arranged controls, can be difficult to use the method to select controls rodeo individuals who want to align. In this case, click on one of the controls you want. Then hold down the CTRL key and click on the other you want to add to the group. (Clicking on the currently selected control will delete the rest of the group).
For example, suppose you have three or four groups of texts and edit controls all in the same row. Select the entire row and uses "Layout - Align Buttom" (alignment of buttons) to arrange the buttons in a straight line, which is flush with the bottom.
It is important to emphasize the effect of the groups arranged vertically with respect to the horizontal. If you select a group of three buttons next to each other in a horizontal row and uses left alignment, the editor will use the button that is further to the left as a guide and move the other buttons so that the left edge of the controls be aligned.You have three buttons, one on top of another! (Use the command Edit - Undo (undo) and start over again).

7.9.2 Centered

This group of commands can be used to focus one or more controls horizontally or vertically in the dialogue box. Note that a group of multiple controls is considered a single unit and be centered. The space between the individual controls will not be affected.

7.9.3 Using the Same Size

After adding a group of similar controls, to make the appearance of the dialog box controls could be improved if they had the same size. In this case, use the command "Layout - Make Same Size" (make the same size) to resize the selected group and make it the same size as the control is in the top left.
If you want the new size is one of the other controls in the group, then move this particular control to the top left corner temporarily. After making all the controls of the same size, re-position it in the right place.

7.9.4 Tab order

Command-Tab Layout order (sort of sequence that follows the focus) can be used to change the order in which the Tab key to move between make controls when the dialog box is displayed. By default, the tab order is the order in which the controls have been progressively added. Because the first two of a new dialog controls are always OK and Cancel buttons, these will come first in the tab order.
To change the order, select the "Layout-Tab order" menu. By default, each interactive control screen will appear with a small window in the upper left corner. Number appears in each window that indicates the position in the sequence control of the focus.
To change the sequence, click on each window starting with the control you want to take the first place in the sequence of focus. If you want to leave control in the same order, you must click on the window until I get the proper position. If you get confused just go out and start again. Exit To enter any keyboard character. (Press the ESC key to exit this mode).
If you want to include text and group controls in the tab order, set the menu command before selecting the command Layout - Tab. Although a text control or group can not be passed with the Tab, this feature allows you to use these controls to establish a mnemonic (shortcut) for interactive control that has no label associated with it. Simply give the text control mnemonic value (make precede the character within the text with a amspersand) and the focus will move to the next edit control (in the tab order) when the mnemonic is pressed.

7.9.5 Returning to create a sequence of IDs

Each control you create is given a sequential ID value beginning with the first number in the range assigned to the specific type of control. For example, the edit controls have IDs between 100 and 199.
Imagine that you have created three editors with IDs 101, 102, and 103. If you delete the control with the ID of 102 the following is assigned the value 104. By the time you finish designing the dialogue, you will discover that the values are out of sequence. While this has no effect on the mode of dialogue is much easier to make changes to a box that has all of its visual checks ID sorted.
The command "Layout - resequence IDs" (sequences Recreate IDs) can be used to rebuild the IDs for individual types or control for all controls based on their order of Tab. When do the sequence again IDs, the first such control in the tab order will be marked the first available ID value (within range) and the next control in the tab order will be assigned the next ID, etc.

The range of values for different controls IDs are as follows:

CONTROL	Nº DE IDENTIFICACIÓN
Edit	100-199
Command buttons	200-299
Listboxes	300-399
Scrollbars	400-499
Labels	500-599
Frames	600-699
Comboboxes	700-799
Chexkboxes	800-899
Radio buttons	900-999

7.9.6 Screen Resolution

This command can be used to create a dialogue for one of the following resolutions:
• 640x480
• 800x600
• 1024x768
This feature allows you to design dialogues on a monitor that has a higher resolution than what your client may be using. For example, if you are using a resolution of 1024x768, the dialog you are designing will not work on a system that is running at a resolution of 640x480. To avoid a possible problem, use the command "Layout - Display Resolution" and make your selection to redesign the dialog editor window according to the resolution.
If your application uses a display format with a toolbar and / or status bar, then make sure the editor window also has the main toolbar and the status bar (status). Otherwise, all formats may not fit in the application window.

7.9.7 Radio buttons commands

The menu commands are available to make or break a group of radio buttons. When you have defined a set of buttons, you may find it easier to use the command "Layout - Undo all Radio Groups" (undo all radio groups) and return them if you find that you do not operate as expected. (Refer to section Radio button controls for more information on how to use these commands).

7.9.8 Grid Size (Resolution)

The dialog window has a mesh size of five pixels by default. This means that a control can be moved or resized five pixels at the same time so you can see the effect. Although it is recommended that this value does not decrease, this size can be increased temporarily when aligned controls.
Note that this command is only effective when you move or resize controls with the arrow keys. Mouse movements are always of + / - 5 pixels.

7.10 Runtime functions

You can use a set of functions in the areas for specific code into the source file programmer BASIC and can thus change the appearance and the operation of the controls in the dialog. In all cases:
HDlg% refers to an integer variable used as dialog handler. HDlg% begin with the prefix name characters dialogue. (In the dialogue example uses the variable dlgCETHDLG%).
% ID is a decimal value that identifies a specific control. These numbers are within a certain range depending on the type of control.
The use and syntax of the functions currently available are grouped here by the kind of control which affect them.

7.10.1 Checking Box Controls

cGettCheck (hDlg% ID%, ADDROF (STATE%))
Gets the state of the control specified by ID% and returns within the variable STATE%. If the control is "proven" this value is 1, otherwise it is 0.
cSetCheck (hDlg% ID% STATE%)
Sets the state of a check box specified by ID% within the variable STATE%. If STATE% is 1, the check box is initially checked. If the value is 0, then you are as unproven.

7.10.2 Combo box controls

cAddComboContents (hDlg% ID%, MATADDROF (ITEMS $))
cComboAddString (hDlg% ID%, STR $)

cComboDelString (hDlg% ID%, INDX%)
cComboInsString (hDlg% ID%, STR $, INDX%)
For a description of these functions, refer to the listings box functions with the same generic name.
cComboSetChoice (hDlg% ID%, STR $)
Sets the default selection for combo box specified alphanumeric STR $.
cGetCombo (hDlg% ID%, ADDROF (STR $))
Take user selection combo box specified by ID% and introduced into STR $. This String is empty if no selection has been made.
cLimitComboText (hDlg% ID% MAX%)
Sets the maximum number of characters that can be entered in the combo box editor specified MAX% value.
cSetCaret (hDlg% ID% POS%)
Sets the cursor position to the associate editor combo box indicated the value of POS%. The special value POS% = 0 places the cursor before the first position. If the value of POS% is greater than the number of characters of the editor will be positioned at the end of the editor.
Note that this feature should be used with Event subroutine. If the cursor is currently located above the control specified by ID%, the result is unpredictable.

7.10.3 Command buttons controls

cDefaultBtn (hDlg% ID%, NEWDEFAULT%)
Changes the current default command button specified by ID%, the new value specified by the button ID NEWDEFAULT%.

7.10.4 Edit controls

cGetEditText (hDlg% ID%, ADDROF (STR $))
Gets the current value of the edit control specified by ID% and enters it into the variable $ STR.
cGetLimitEditText (hDlg% ID%, addrof (MAX%))
Used to get the maximum number of characters that can be entered into a text editor control.
cLimitEditText (hDlg% ID% MAX%)
Limits the number of characters that can be introduced into specific control.
cSetLimitEditText (hDlg% ID% MAX%)
This function is similar to cLimitEditText.
cSetCaret (hDlg% ID% POS%)
Sets the cursor to the text editor POS% value. The special value POS% = 0 places the cursor before the first position. If the value of POS% is greater than the number of characters of the editor will be positioned at the end of the editor.
Note that this feature should be used with Event subroutine. If the cursor is currently located above the control specified by ID%, the result is unpredictable.
cSetEditText (hDlg% ID%, STR $)
Sets the contents of the edit control specified by ID% to variable $ str.
cShowEditWrap (hDlg% ID%, FLAG%)
Sets line breaks within a line edit control multiple on or off. When FLAG% = 1 is added at the end of each line two carriage returns followed by an end of line for each line that has been broken due to the rupture of word. If FLAG is 0%, the words that have been chipped or cut into the next line will appear as a single line.

7.10.5 List box controls

cAddLBoxContents (hDlg% ID%, MATADDROF (STR $))
Sets the current value of the control box% ID list specified at values that are within the STR $ matrix. This matrix can contain up to 15000 elements.
cLBoxAddString (hDlg% ID%, STR $)
Add the contents of STR $ alphanumeric end of the list box. The items will be reclassified if the control is set for the classification property.
cLBoxDelString (hDlg% ID%, INDX%)
Clears the value Alphanumeric index INDX% list box.
cLBoxInsString (hDlg% ID%, STR $, INDX%)
Insert new element alphanumeric before the index value INDX%. Put INDX% to -1 if you want to add the final alphanumeric list box. In any case, there shall be no classification.
cGetLBoxSelection (hDlg% ID%, MATADDROF (RESULTS%))
Gets the current selection box control specified by ID% listings and enter index values within the entire matrix RESULTS%. Because the number of selected elements in the array is always within the element 1, we can speak of him as the counter field. Thus the values selected alphanumeric indices start from item 2.

If the list box selection is simple (the default), the array is sized to 2. When selecting one of the selection elements, the first element of the array will have value 1, and the selected element index value stored in the second element of the array.
When using a multiple selection list box, the size of the array is determined by the number of items selected. Your program will need to loop counter based on the field (The array element 1) to find the index values of each of the selected items.
If no selection is made, the first element of the array will have zero value and the second element is empty no matter what type of list box being used.
cLBoxSetSel (hDlg% ID% SEL%)
Sets as selected text (highlighted) to the index value SEL% and this will be the last visible text in the dialog box.
cGetLBoxString (hDlg% ID% INDEX% ADDROF (STR $))
Get selected text associated with the index value INDEX% of list box control specified by ID% and return it to the variable $ STR.
cLBoxTabStops (hDlg% ID% STOP%)
Sets tab stops every STOP% characters.
cLBoxTopIndex (hDlg% ID%, TOP%)
Sets the first visible item list box to TOP% rate. If you are also using a call to cLBoxSetSel, make sure you make a call to cLBoxTopIndex after cLBoxSetSel changes because the higher value of the index.

7.10.6 Radio buttons controls

cGetCheck (hDlg% ID%, ADDROF (STATE%))
Gets the state of the control specified by ID% and introduces it into the variable STATE%. If the control is "proved" the value is 1, otherwise it is 0.
cGetRadioGroup (hDlg% ID%, ADDROF (SEL%))
Returns the ID of the currently selected radio button from the group with the specified ID% and enter it into the variable SEL%.
The return value will be zero if no button is not checked. This will be possible only when the user has not made any selection and not set the initial state property "checked" for one of the buttons in the group.
cSetCheck (hDlg% ID% STATE%)
Sets the state control with the ID value into the variable% STATE%. If STATE% equals 1, will initially be tested. If the value is 0 then the button can not be proven. No other button in the group will be affected by this function.
The general purpose cSetCtrlValue routine can be used to set the state of a button checked while checking remove any button that was previously checked.

7.10.7 Scrollbars controls

cGetScrollPos (hDlg% ID%, ADDROF (POS%))
Returns the current position of the scroll bar within the variable POS%.
cSetScrollPos (hDlg% ID% POS%)
Sets the scroll bar at position POS%.
cSetScrollPage (hDlg% ID% PAGE%)
Sets the value of the "page" to the scroll bar control to PAGE%. The default value is 1/10 of the range. Assuming that the range was 100, each rectangle movement equivalent to + / - 10 and seem to move in 1/10 of the length of the scroll bar. To set a page value 1/20, set the value equal to the rank PAGE% divided by 20.
cSetScrollRange (hDlg% ID% MIN% MAX%)
Sets the maximum and minimum values of MIN and MAX%%.

7.10.8 General routines

cAppTitle (hDlg%, TITLE $)
Sets the title of a display format to the value of $ TITLE
cDlgTitle (hDlg%, TITLE $)
Sets the dialog title to the value of $ TITLE.
cEnableCtrl (hDlg% ID% STATE%)
Active (STATE% = 1) or off (STATE% = 0) control. A control when offline is grayed out and can not get the input focus.
cEndDlg (hDlg% CMDID%)
Calls the routine (file. HDI) ending dialogue with the identifier of the control that generated this event (end of the dialogue). This operation is usually done when you press a button that has enabled the property to dialogue. PROGRESS In the sample program at the end of chapter use this feature.

cEndForm (hDlg%)
Ends the specified display format.). This operation is usually done when you press a button that has enabled the property to dialogue. See the code that handles the button press 'Exit' (case 204) INVOICE example program, which is documented at the end of the chapter.
cFormBtnClick (VAL%)
Send an event defined by the user as the click of a button to the currently active form. For example, you can have a menu item that invokes a new format. If the user selects an item, this function can be called from handler ID_WINSUB (cetwin.b is inside, which is the default) to tell the active format to save and exit (with CMDID% VAL% made BtnClick within the subroutine). If there is no active form, the call is ignored.
cGetCtrlText (hDlg% ID%, ADDROF ($ TEXT))
Gets the value of a static text. Text areas are the contents of static text controls, the title of a control group, and tag command buttons, check boxes and radio buttons. An empty String is returned for any other control.
cGetCtrlValue (hDlg% ID%, ADDROF (STR $))
Find content and introduces interactive controls on STR $. This routine returns all values as alphanumeric. The command buttons, text and controls group always return an empty String. The return values for other controls are:
Returns Control Type
Edit control content.
The check boxes selected state is well "1" (checked) or "0" (not verified)
Radio buttons selected status is good "1" (checked) or "0" (not verified)
Scroll Bars The numerical value of the position as a String.
Combo Box User selection.
List Boxes The selected alphanumeric.
Note that the alphanumeric is returned list boxes instead of index value that is returned when using specific routines list boxes. Note also that in a multi-select list box control will return only the first alphanumeric. You must use the function to search alphanumeric cGetLBoxSelection selected.
cSendBtnClick (hDlg% VAL%)
Send the click of a button with the subroutine BtnClick made CMDID% VAL%. For example, you may want to double-click event of a list box to complete the dialogue (as if the user had selected an item and then had punctured the OK button). In this case, add code to the event subroutine to send a button click (for example if the OK button has the ID 200, put 200 VAL%). Note that this function can not be used from the Init subroutine since dialogue is not active at this time.
See the example PROGRESS program at the end of this chapter to see how you can display a dialogue without requiring any user interaction.
cSetCtrlText (hDlg% ID%, $ TEXT)
Sets or changes the value of a static text control with the specified ID value to a new alphanumeric characters. Text areas are static content text controls, the title of a control group, and the label for command buttons, check box and radio buttons. The call has no effect on other types of control.
cSetCtrlValue (hDlg% ID%, STR $)
Sets the content of interactive control STR $. This routine expects all values are alphanumeric (for example, if a check box must be checked, pass "1" and if not "0").
Other controls can be set as follows:
Control Type Value
The contents of an edition alphanumeric
The state checkboxes, either "1" (checked) or "0" (not verified)
Radio buttons status, either "1" (checked) or "0" (not verified)
Scroll Bars The numerical value of the current position of a String.
Combo box default selection.
cSetFocus (hDlg% ID%)
Sets input focus to the control with the specified ID%.
cShowCtrl (hDlg% ID% STATE%)
Shows or hides the control with the specified ID value. If STATE% is 1, the control will appear. The control will be hidden (and off) is zero if STATE%.

7.11 Dialog source files

Dialogues Editor generates the necessary code to display the controls according to how their properties are defined, the code detects events as the click of button and returns the values of interactive controls.
Two files are created automatically when you save a dialogue or a display format. These are:
Function File

prefix.idh The file header contains all the code that represents and dialogue processes. Not modify the file. Idh, otherwise the editor will be unable to load the dialog.
By default, each interactive control is defined as global within this file. The names of the global variables are created by concatenating the prefix of dialogue with the BASIC variable name. The sentences are not sensitive to case-sensitive, so you can mix them in order to increase the readability of the variable names. In dlgCET, which causes the button to close the dialog and is defined as:
$ GLOBAL dlgCETENDBTN%
Prefix.b The source file is the dialogue that you can modify in order to process the events that occur in the dialogue (clicking a button, selecting an item in a list box, ..) according to the particular application.
It is important to note that this file does not contain information about the visual characteristics of the dialogue, that is, information that serves to represent the dialogue or to establish its mode of operation. Therefore, you could open the box (the. HDI) and make changes while you open the source file (the. B) in another editor window.
The only source file lines that belong to a specific dialogue are the names of BASIC subroutines. So whenever the prefix name of dialogue, dialogue must update the editor using the command 'File-Rewrite. HDI and. B' in order to restore the balance between the two files dialogue.
For general information about using BASIC subroutines, see Subroutines BASIC chapter of this manual.
The source file of the dialogue is organized into the following parts:
Includes top file extension. Idh using the editor (with a # include statement). If ownership of the dialogue 'not make the controlling variables $ GLOBAL' (Global variables do not BASIC controls), then the variables you want to use BASIC and global must be defined here before the # include.
Make a sentence main subroutine GOSUB to call routines that create and show the dialog. These routines are stored in the file. HDI.
Click Button subroutine processes the button click event.
Process Event Handling several special events, which occur as a result of a change in the value or state of a control.
Initialization subroutine Sets or changes the initial state list boxes and other controls.
The code produced by such dialogue (dlgcet.b) is used to explain the individual parts of the program.
CETDEMO, the main program that calls the dialog dlgcet.b has also been included in the manual. Other programs are also documented examples at the end of this chapter.

7.11.1 Program upper part

The dialog editor uses the character "REM @ #" to mark special lines in the source file. One of the first lines identifies the publisher and the publisher's version that was used to create this dialogue.
The sentences REM @ # define current dialog controls and screen display are stored in a file. Idh (Include Dialog Header) that is specified by an # include statement. By default, the file also contains the # include code to return the values of interactive controls. And $ GLOBAL statements that define the shared variables.
At the top of the file will find the following statements dlgCET.b:

```
REM@#       Dialog dlgCET V2.3
REM              first dialog save area
REM@# 1ststart
REM          The Next button allows you to change the contents of the
REM           list box.  The following statements define the global variables
REM           that are used in processing the list boxes.
$GLOBAL MAT FIRST$, MAT SECOND$, MAT MULTI$
$GLOBAL USESECOND%
REM@# 1stend
#include "Dlgcet.IDH"
```

Note the lines that appear above the # include statement. These statements REM # mark the first safe area where you can add user code. Any code added beyond the limits established by these brands, not saved for the next time you load the dialog in the dialog editor.
Since all interactive controls dlgCET dialogue (our example) are used to pass information to the main program (CETDEMO), we used the default option, which is to let the editor automatically generates all the sentences of all $ GLOBAL interactive controls (in the file. HDI). That way you only have to define extra variables to use as global, in this case, we have used an array of strings in the list box, and a variable for processing the 'Next' button:

```
REM@# 1ststart
$GLOBAL MAT FIRST$, MAT SECOND$, MAT MULTI$
$GLOBAL USESECOND%
```

```
REM@# 1stend
```

If ownership of the dialogue 'NOT make $ GLOBAL variables Control "was used (in this example does not use), then you must define all variables BASIC interactive controls that would be used as global variables within this secure area. (It has been done this way in the program INVOICE example, since very few were used as global variables, this was way better than the default).
All $ global variables must be defined before the # include. Because at that point it will read the file. Idh to create dialogue. Otherwise, it would generate a compilation error. Note also, that if the same variable is defined first as $ GLOBAL and later as COMMON, the compiler will detect a duplicate symbol status and display a notification message, it might not be a fatal error.
The dialog editor w/32 creates four BASIC subroutines for each dialog. The value specified in the property dialogue prefix (prefix) is used as the name of the first subroutine. The names of the other three subroutines begin with the same value. In our example the prefix property value is "dlgCET", and therefore the names of the four subroutines will:

```
$SUB dlgCET
$SUB dlgCETBtnClick
$SUB dlgCETEvent
$SUB dlgCETInit
```

Each of these subroutines are described below.

7.11.2 Prefix subroutine

This is the subroutine that you need to call from your main program (the one containing the statement $ MAIN) whenever you want to show dialog designed.
In our example, the main program (CETDEMO) invokes the dialog with the following statement. Remember that the CALL is sensitive to case-sensitive, so if you used DLGCET this statement cause an error.
CALL dlgCET
When calling this subroutine will create and show the dialog running using the following code:
$ SUB dlgCET
REM @ # dlgstart
REM @ # dlgend
GOSUB dlgCET.CREATE
$ EXIT
Note that you have a safe area (the space between the comments dlgstart REM and REM @ # @ # dlgend) to specify the code you want to run before the dialog is created (and before you run the subroutine dlgCETInit initialization dialog). In the example INVOICE, this area is used to size the array of items in the invoice lines.
THE GOSUB statement transfers control to the routine dlgCET.CREATE, which is stored in the file. HDI. Since there are other initialization subroutine calls control, event detection, display and close dialogue.
The judgments of the beginning of the subroutine dlgCET.CREATE define the number of controls (within the global variable prefixCTRLS.CNT%), and dimensions the array of control IDs (IDs) (this is the global variable $ prefixCRTLS) and corresponding properties to be defined by the user (will be stored in the global variable $ prefixPROPS). The array of controls contain ID values in the order of focus.
REM Create the dialog
REM Add the controls and initialize the ctrls% and PROPS $ arrays
dlgCET.CREATE:

REM The controls array
dlgCETCTRLS.CNT% = 16
DIM dlgCETCTRLS% (dlgCETCTRLS.CNT%)
DIM dlgCETPROPS $ (dlgCETCTRLS.CNT%, 10)
Suppose, for example, that one of the properties of interactive controls to be defined by the user, you have decided that you will use to define the type of input that is required (for example a positive integer, or an integer that must be in the range 1-999). In that case, when the user clicks a button to course entry, you might look at the array of $ prefixPROPS properties to see if the data follows the correct format to be followed by that entry.
See the example program VALIDATE, we explained at the end of this chapter to look at how you can use the properties to be defined by the user in a real application.

7.11.3 prefixBtnClick subroutine

The second subroutine is named dlgCETBtnClick is called whenever a button is selected. The global variable CMDID% have the ID of that button.

Code is generated automatically (in the file. HDI) to process the OK and Cancel or the Close command on the System menu. If you want these buttons you to realize additional tasks, add custom code to process pressing those buttons within the reserved area between REM and REM btnstart @ # @ # btnend.
In dialogue dlgcet.b example, the following code is added to process Browse and Next buttons. Because the check boxes are special types of buttons, disable the check box is also processed here. We use comments to describe the operations. The syntax of the CALL statement is explained in section RunTime functions later.

```
REM@# btnstart
SELECT CMDID%
CASE 202
    REM The Browse button was selected to display a File Open
    REM  browser and put the selected file into the edit control.
    REM  Change the working directory for the open browser to
    REM  C:\WINDOWS and restore it afterwards.
    CALL Bgetcwd ( ADDROF (savdir$) )
    CALL Bchdir ( "C:\WINDOWS" )
    CALL cOpenBrowser ( ADDROF (SELECTION$), "" )
    CALL Bchdir ( savdir$ )
    IF SELECTION$ <> ""
            REM A file selection was made.
            CALL cSetEditText ( dlgCETHDLG%, 101, SELECTION$ )
    IFEND
CASE 800
    REM  Use the value of the Disable check box to enable/disable the
    REM  option check boxes.
    CALL cGetCheck ( dlgCETHDLG%, 800, ADDROF (STATE%) )
    IF STATE% = 1
            REM Box is checked - disable controls.
            CALL cEnableCtrl ( dlgCETHDLG%, 801, 0 )
            CALL cEnableCtrl ( dlgCETHDLG%, 802, 0 )
    ELSE
            REM Box is not checked - enable controls.
            CALL cEnableCtrl ( dlgCETHDLG%, 801, 1 )
            CALL cEnableCtrl ( dlgCETHDLG%, 802, 1 )
    IFEND
CASE 207
REM  The Next button may be used to change the contents of the
REM  list box.  The $globals for the second list are defined at the
REM  top of the file.
    IF USESECOND% = 1
            MSGBOX "Already displaying second choices"
    ELSE
            CALL cAddLBoxContents (dlgCETHDLG%,300, mataddrof
 (SECOND$))
            USESECOND% = 1
    IFEND
CEND
REM@# btnend
```

Note that none of the buttons above will lead to the end of the dialogue (with $ EXIT). A dialogue ends when the operator selects a command button that has the property "final" dialog on (as the default OK button and Cancel button). Press the ESC key has the same effect as pressing the Cancel button. There is a special case when the operator terminates the dialog by selecting the menu command Close dialogue system. Then, return the value 1.
A update format can detect that the system is closed (close), has left the program (Exit), or you pressed Alt + F4, in Case 1 SELECT CASE your subroutine handling mouse events.
Code exists in the file. Idh used to detect a condition to dialogue, and assigns the ID button end condition within dlgCETENDBTN% variable and returns the values of all interactive controls. Since these controls are defined as global, this information will be passed to the main program ($ MAIN) for further processing.
The generated code in the file. HDI has been included here:

```
DlgCETENDDLG0:
```

```
call cGetCheck ( dlgCETHDLG%, 800, addrof (dlgCETDISABLE%))
call cGetCheck ( dlgCETHDLG%, 801, addrof (dlgCETOPT1%))
call cGetCheck ( dlgCETHDLG%, 802, addrof (dlgCETOPT2%))
call cGetEditText ( dlgCETHDLG%, 101, addrof (dlgCETFILENAME$))
call cGetLBoxSelection (dlgCETHDLG%,300,mataddrof (dlgCETSINGLE%))
call cGetLBoxSelection (dlgCETHDLG%, 301,mataddrof (dlgCETMULTI%))
dlgCETENDBTN% = CMDID%
call cEndDlg ( dlgCETHDLG%, CMDID% )
RETURN
```

When creating dialog boxes, often find it convenient to write the code editor to find the control values. Furthermore, when creating display formats typically write your own code to get these values, depending on your specific needs. In this case, the property may establish dialogue so that these lines are not generated unnecessarily. (This is the way it was done in the example the end of chapter INVOICE).

7.11.4 PrefixEvent subroutine

This is a general purpose subroutine is called every time there is a specific type of event. For example, every time a program detects a change in an edit control, you call this subroutine so you can validate and format user input.
This subroutine is a SELECT-CASE statement that process can be modified for special events. The sample program does not use this feature. Thus, no extra code is included here. (See the example programs at the end of this chapter).

```
$ SUB dlgCETEvent
SELECT CET_EVTYPE%
REM @ # evtstart
REM @ # evtend
CEND
$ EXIT
```

These are global variables that are used to process the events:
Variable Value
CMDID% The ID of the control that generated the event.
CET_EVPARM1% depending on the type of event, the contents of this variable will be the new position of the scroll bar, the next control that will be the focus, or the current cursor position.
CET_EVTYPE% The code indicating the type of event.
Recognizes the following events:
Exchange Event
1 An edit control has changed or because they entered a character or because he has made a cut or copied to a new value. In this case the variable CET_EVPARM1% should have the cursor position.
2 A list box control has changed to a new selected item to the box with a simple click.
3 A list box control has changed when selecting a new item in the box with a double click.
4 The position of the scroll bar is changed. The new position is passed in the variable CET_EVPARM1%.
5 An edit control Combo box changed. (See the event 1).
6 A control has received the input focus. The operator selected the new control using a mnemonic, the Tab key or clicking the mouse.
7 A control input has lost focus (because the user selected another control). The next ID receive the focus control is passed through CET_EVPARM1%. This ID value will be zero if the following control is not part of the dialogue. (The user could have triggered another application by clicking its window). If you are validating input fields, you should always ensure that the following control is a Cancel button before notifying the user of an invalid entry.
8 It has made a selection of Combo box with a simple mouse click.
9 The list of a combo box is displayed. This event is useful when you want the user to enter a value or enter it in the list (down), or find an existing one. In that case, you can check the value entered by the user, according to this value, alter the elements of the dropdown combo box.
Because when any changes occur in an edit control generates an event number # 1 at a time, you can add code to check and format the data at that time. In that case, the global variable contains the value CMDID% ID of the edit control. In our example, the following function gets the current value of the multiline edit control within dlgCET.by returns to the variable $ dlgCETFILENAME:

```
cGetEditText call (dlgCETHDLG%, 101, ADDROF (dlgCETFILENAME $))
```

This feature allows you to validate the contents of a control character by character without waiting for the user to select another control. For example, suppose you create a dialog with an edit control that seeks entry

into a specific format. If the user types as alphanumeric "ABC", will be made three calls to this subroutine, one for each character.
This method can also be used to check the first character of the input and dynamically change always capitalized.
The program uses a procedure example INVOICE to process the data in the edit control after they enter the field (and therefore has lost the focus):
1. Use the flag 'modified' to know that an event has occurred of type 1 (an event editing control has changed).
2. Show the flag 'modified' and (conditionally) executes its validation routine when data is inserted completely, giving an event of type 7 (control has lost focus).

7.11.5 PrefixInit subroutine

The last subroutine is called at the time the dialog is initialized (before it appears on screen). This is where you can set or change the initial state of any control.
If you have defined a list box, the initial elements should be added here. The code below has been caught dlgcet.b example dialogue. Alphanumeric matrices that should appear in the two boxes listed are defined as local variables and a call is made to cAddLBoxContents to set the current value of the checks for arrays. (Values of alphanumeric or texts are set in the program CETDEMO).
Note that a call is made to the box cSetCheck "Option 1" is checked or checked when the dialog appears.

```
$ SUB dlgCETInit
REM @ # inistart
REM Initializes the contents of both boxes listed.
CALL cAddLBoxContents (dlgCETHDLG%, 300, mataddrof (FIRST $))
CALL cAddLBoxContents (dlgCETHDLG%, 301, mataddrof (MULTI $))
REM Set the initial state of the check box Option 1
REM found.
CALL cSetCheck (dlgCETHDLG%, 801, 1)
REM @ # iniend
$ EXIT
```

At the end of this subroutine is a special secured area which can be used to store any user routine (GOSUB) or user role. This feature is used in many programs examples (INVOICE).
It is also important to note that there has been no error handling in this example. But in a real application, each BASIC subroutine that could result in error should have its own error handler using the sentences "ON ERROR GOTO". In that case, it may store routines errors in the previously cited last safe area.

7.12 Copying a dialog

A program is a subroutine BASIC dialogue and sentences containing a $ SUB. The following example shows how to use OBW32 to compile individual modules in separate files. Obj and then link them together.

```
OBW32-c dlgcet.b
Cetdemo.b cetdemo OBW32-or dlgcet.obj
```

This is done automatically when using W32APBLD, utility w/32 integrated development environment to create and develop projects that will result in executable programs. Projects have been developed for the program menu CETDEMO examples and examples of other programs.

7.13 The main program

The program displays the dialog CETDEMO example dlgCET.b example that has been used to illustrate some of the features explained in this chapter. In this example, the information entered in the dialogue, is passed to the main program, and there is printed on the screen.
The program is fully explained cetdemo.b and so we have included below:

```
$MAIN

REM Un programa ejemplo que saca por pantalla una caja de diálogo
REM multifunción.
REM Las siguientes variables globales definen los controles en el
REM diálogo.
$GLOBAL dlgCETENDBTN%
$GLOBAL dlgCETFILENAME$
$GLOBAL dlgCETOPT1%
$GLOBAL dlgCETOPT2%
$GLOBAL dlgCETDISABLE%
```

```
$GLOBAL MAT dlgCETSINGLE%
$GLOBAL MAT dlgCETMULTI%

REM Las siguientes variables globales están definidas para pasar el
REM contenido de la caja
REM de listado al diálogo y permitir al programa determinar qué conjunto REM de
alfanuméricos
REM a saca por pantalla en la caja de listado de selección sencilla .
$GLOBAL MAT FIRST$
$GLOBAL MAT SECOND$
$GLOBAL MAT MULTI$
$GLOBAL USESECOND%

REM Dimensiona la matriz de caja de listado.
DIM FIRST$ (10), SECOND$ (3), MULTI$ (10)

REM Inicializa el flag de la segunda matriz a falso e inicializa los
REM elementos de la caja de
REM listado.
USESECOND% = 0
FIRST$ (1) = "This is selection 1"  \  FIRST$ (2) = "This is selection 2"
FIRST$ (3) = "This is selection 3"  \  FIRST$ (4) = "This is selection 4"
FIRST$ (5) = "This is selection 5"  \  FIRST$ (6) = "This is selection 6"
FIRST$ (7) = "This is selection 7"  \  FIRST$ (8) = "This is selection 8"
FIRST$ (9) = "This is selection 9"  \  FIRST$ (10) = "This is selection 10"
SECOND$ (1) = "This is the second selection 1"
SECOND$ (2) = "This is the second selection 2"
SECOND$ (3) = "This is the second selection 3"
MULTI$ (1) = "This is multi selection 1"
MULTI$ (2) = "This is multi selection 2"
MULTI$ (3) = "This is multi selection 3"
MULTI$ (4) = "This is multi selection 4"
MULTI$ (5) = "This is multi selection 5"
MULTI$ (6) = "This is multi selection 6"
MULTI$ (7) = "This is multi selection 7"
MULTI$ (8) = "This is multi selection 8"
MULTI$ (9) = "This is multi selection 9"
MULTI$ (10) = "This is multi selection 10"

REM Ahora invoca el diálogo llamando a la subrutina BASIC.
CALL dlgCET

REM Chequea para ver si el diálogo fue cancelado y saca por pantalla el
REM mensaje
REM apropiado. Un diálogo cancelado devolverá bien ID=1 si el diálogo
REM fue cerrado
REM del menú de sistema o ID=201 cuando el botón de cancelar fue
REM seleccionado
IF dlgCETENDBTN% = 1
    MSGBOX "Dialog canceled from the system menu!"
    QUIT
ELSE
    IF dlgCETENDBTN% = 201
            MSGBOX "Dialog canceled with the Cancel button!"
            QUIT
            IFEND
IFEND

REM Saca los resultados del diálogo por consola.
```

```
PRINT
PRINT "The state of the disable... check box was: "; dlgCETDISABLE%
PRINT
PRINT "The contents of the filename edit control is: "; dlgCETFILENAME$
PRINT

REM Saca por pantalla la caja de listado de selección sencilla.
REM Primero comprueba  si se ha hecho una selección.
REM El contador será 0 si no existe  selección.

IF dlgCETSINGLE% (1) <> 0
    IF USESECOND% = 1
            REM Obtiene el elemento 2 de la segunda matriz.
            PRINT "The single selected item was: "; SECOND$
 (dlgCETSINGLE% (2))
    ELSE
            REM Obtiene el elemento 2 de la primera matriz.
            PRINT "The single selected item was: "; FIRST$
 (dlgCETSINGLE% (2))
    IFEND
IFEND
REM Saca por pantalla las opciones de la caja de listado de
REM miltiselección.
PRINT
PRINT "The number of strings selected in the multiple selection list box was: "; dlgCETMULTI%
(1)
FOR I% = 2 TO dlgCETMULTI% (1) + 1
    PRINT "     "; MULTI$ (dlgCETMULTI% (I%))
    NEXT I%
WAIT
END
```

7.14INVOICE - An example using visual format

INVOICE program illustrates how the display format can be used to replace a window of text-based application. The source files for this program are:
Using multiple product characteristics in this sample program. INVOICE uses a combo box and several buttons that are used to create entries in an array of bills, searching existing invoices, change data, etc.. It also has code to validate and format the data entered by the user, and to archive data, cancel operations, or exit the program.
W32APBLD development environment, has been used to create the project invoice.prj. Recommend charging this project within the development environment, run the project, and then see the same code. Sources for this project are:
invmain.b The source code for the main program
dlginv.b The source file display format (dlginv.idh), which shows the input screen, formats and validates user input, calls a dialog box that gets the items of the invoice lines and running all operations of input and output file.
dlgline.b The dialogue source file (dlgline.idh) that gets the items of the invoice lines.

All these files are automatically installed with your version of CETWIN in the default directory \ cetlib \ samples. This software is also included in a file called facturas.zip within our BBS in the library windows.
The program INVOICE framework uses the default WINDOWS CET programs, so there are no menus or toolbars special. There has been a change to the file invmain.b (cetwin.b file copy) to add a line that sets the background color to white, so you're not seeing a white window on-black before the initialization process . (We could also have used the environment variables of background color and foreground file W32APBLD.w32 environment variables).

7.14.1 The main program

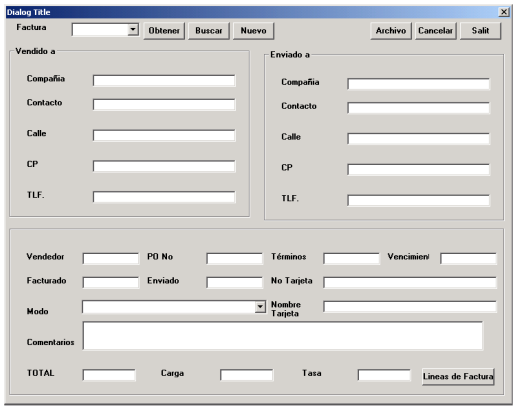
The main program, invmain.b, declares global variables that have passed between subroutines in a display format, calls a function that prints out the title of the display format, and calls INV1, which is the main subroutine display format, dlginv.b stored in the file, and exit.

```
$ MAIN
$ GLOBAL mat $ rec
$ GLOBAL docModified%
$ GLOBAL newFlag%
$ GLOBAL dataEntered%
cAppTitle call ("Sample App Builder w/32 Program - INVOICE")
call INV1
call cForceExit
quit
```

Although you may want to open and close the data files you need from within the main program, we suggest that any other process is done in the source file for the format. In another case, a flash appear each time you stop to re-format the text-based Windows application main program.

7.14.2 The visual format source file

The format stored in dlginv.b uses the following input screen for the invoice information (invoice).
Ownership of the dialogue 'not make the controls global "(global controls do not) is enabled for all
display format processing was done from within your source file using the values of ID checks. It took only a few global variables to pass information to and from subroutines format source file (dlginv.b) and the source file of invoice lines (dlgline.b) and the main program. All these statements $ GLOBAL defined before the # include.




```
REM@#       Dialog inv1 V2.3
REM@# 1ststart
$GLOBAL mat rec$            \ REM the invoice record array
$GLOBAL docModified%        \ REM used to check changes in record
$GLOBAL newFlag%                    \ REM used in the line item dialog
$GLOBAL dataEntered%        \ REM used in the line item dialog
REM@# 1stend
#include "inv1.IDH"
```

If all variables were global control (using the default), no unnecessary lines only have $ GLOBAL statement in the file. HDI, but also those sentences would also have to be duplicated in the main program.

```
$ SUB INV1
```

Only added a line of code in the secure area of the main subroutine.

```
REM sized matrix invoice record so you can have 373 REM items
REM (23 items in this format 7x50 REM line elements within the dialog).
dim rec $ (373)

$ SUB inv1BtnClick
```

Added code to this subroutine to handle keypress events command buttons by the customer.
Note that if you use any check box or radio buttons can also be processed in this subroutine.

```
SELECT CMDID%
CASE 200
    REM Botón de búsqueda
```

```
        msgbox "not implemented"
        $exit
    CASE 201
        REM Botón de elementos de la línea de factura
        REM Invoca el diálogo de elemento de línea.  Si algún elemento de la
        REM línea es actualizado
        REM el diálogo actualizará la matriz del registro de la factura que está
        REM como global y establecerá el flag modificado de documento que REM también una
        variable global.
        call line
        REM Recalcula el total de la factura después de volver del elemento de
        REM la línea perteneciente al diálogo y sacará por pantalla el resultado.
        gosub calc.total
        call cSetEditText ( inv1HDLG%, 120, total$)
        $exit
    CASE 202
        REM Botón de fichero
        REM Toma todos los valores y escribe el registro.
        REM Primeramente toma el número de factura como clave.
        CALL cGetCombo ( inv1HDLG%, 701, ADDROF (key$))
        REM Copia todos los valores de control a la matriz del registro.
        ctrl% = 101
        for i% = 1 to 22
           CALL cGetEditText ( inv1HDLG%, ctrl%, ADDROF (rec$ (i%)))
           ctrl% = ctrl% + 1
        next i%
        CALL cGetCombo ( inv1HDLG%, 700, ADDROF (rec$ (23)))
        REM Escribe la matriz del registro al fichero indexado y limpia
        REM el flag de documento modificado y todos los controles.
        mat write #1,key$: rec$
        docModified% = 0
        gosub clear.ctrls
        REM Si es una nueva factura, añadir el número a la lista de caja combo
        REM y obtiene el siguiente número disponible.
        if newFlag%
            call cComboAddString ( inv1HDLG%, 701, str$ (newInvoice%) )
            newInvoice% = newInvoice% + 1
            newFlag% = 0
            dataEntered% = 0
        ifend
        REM Espera por otra pregunta del número de factura y entonces
        REM va de vuelta al control de factura.
        gosub WaitForInvoice
        call cSetFocus ( inv1HDLG%, 701 )
        $exit
    CASE 203
        REM Botón de cancelación
        REM Cancela la operación actual.  Limpia todos los controles de edición
        REM  y doc modificados y flags de factura nueva.
        gosub clear.ctrls
        docModified% = 0
        newFlag% = 0
        dataEntered% = 0
        REM Espera a la contestación de otro número de factura y entonces
        REM vuelve de nuevo al control de factura.
        gosub WaitForInvoice
        call cSetFocus ( inv1HDLG%, 701 )
        $exit
    CASE 204
```

```
        REM Botón de salida
        REM Detecta cuando ha cambiado un dato y avisa al usuario que
        REM el registro debería ser actualizado o bien cancelada la operación
    REM antes de salir.
        if docModified%
    msgbox "You have a modified record in progress" + chr$ (10) + "Select either File or Cancel
          first"
            $exit
        ifend
        REM Limpia el flag dlg de "exit app" y termina el formato.
        call cForceExit
        call cEndForm ( inv1HDLG% )
        $exit
    CASE 205
        REM Botón nuevo
        REM Crea una nueva factura.  Primero establece el flag modificado del
        REM documento y limpia los controles de edición.  Entonces, copia el
        REM nuevo número de factura al control de combo.
        docModified% = 1
        newFlag% = 1
        gosub clear.ctrls
        call cSetCtrlValue ( inv1HDLG%, 701, STR$ (newInvoice%) )
        REM Establece los valores por defecto para algunos controles.  Estos
        REM pueden también haber sido establecidos en las propiedades de     REMcontrol.
        CALL cSetCtrlValue ( inv1HDLG%, 115, "NET 10" )
        CALL cSetCtrlValue ( inv1HDLG%, 700, "UPS BLUE" )
        REM Inicializa el nuevo registro añadiendo al primer campo
        REM  (el número de líneas) para cada línea.
        mat rec$ =  ( "" )
        num% = 1
        for i% = 24 to 373 step 7
            rec$ (i%) = STR$ (num%) + "."
            num% = num% + 1
        next i%
        REM Llama la rutina para introducir la factura y entonces establece
        REM el focus al primer control de edición.
        gosub EnterInvoice
        call cSetFocus ( inv1HDLG%, 101 )
        $exit
    CASE 206
        REM Toma el botón
        REM Toma un registro existente.  Primero, limpia el flag del documento
        REM modificado y obtiene el número de factura del control combo.
        REM El número de factura es la clave.
        docModified% = 0
        CALL cGetCombo ( inv1HDLG%, 701, ADDROF (key$))
        mat read #1,key$: rec$
        if eof (1)
            msgbox "can't find record"
            call cSetFocus ( inv1HDLG%, 701 )
            $exit
        ifend
    REM Copia los campos del registro a los controles de edición y llama a
    REM una rutina para introducir la factura.  Después de volver, establece
    REM el focus al primer control de edición.
        gosub display.ctrls
        gosub EnterInvoice
        call cSetFocus ( inv1HDLG%, 101 )
        $exit
```

$SUB inv1Event

This subroutine is called when a special event. INVOICE illustrates how code can be added to calculate and validate the input format of a user. When a change in the edit control (CASE 1) activates the flag amendment of Edition. The current process is done when the control has lost its focus (CASE 7).

The sample program VALIDATE explained at the end of the chapter, illustrates another method of data validation and formatting. If you have validation and formatting routines that are common to all programs. In this case, these routines can be stored in a single file and use # include to enter the code during compilation.

```
case 1
    REM Un control de edición a cambiado de tal forma que activa el flag
    REM utilizado para detectar que la factura ha sido modificada.
    docModified% = 1
    REM Establece el flag para comprobar el formateo de campo y calcular
REM el total de factura.
    editModified% = -1
case 5
    REM Un control de combo ha cambiado por tanto cambiamos el flag de          REM
documento.
    REM Solamente nos ocuparemos de "Ship Via" y no la "invoice"  (factura).
    if CMDID% = 700 then docModified% = 1
case 7
    REM Comprueba el flag para ver si un control de edición ha cambiado.
    if editModified%<>0
    REM Efectúa un nuevo formato cuando se introduce datos en un
    REM      control definido como fecha o teléfono en las propiedades
    REM definidas por el usuario.
       i% = 1
       while 1
    REM Busca CTRL% () para que coincida con CMDID%.
              if inv1CTRLS% (i%)=CMDID%
                    if inv1PROPS$ (i%,8) = "DATE"
                              gosub date.format
                              call cSetEditText ( inv1HDLG%, CMDID%, date.fld$)
                    ifend
                    if inv1PROPS$ (i%,8) = "PHONE"
                              gosub phone.format
                              call cSetEditText ( inv1HDLG%, cmdid%, phone.fld$)
                    ifend
              break
              ifend
       i% = i%+1
       wend
    REM Si los campos de carga o impuestos han cambiado inicialice el flag      REM  para
recalcular el total.
       if CMDID%=121 or CMDID%=122
                    gosub calc.total
                    call cSetEditText ( inv1HDLG%, 120, total$)
            ifend
    editModified% = 0
    ifend
case 8
    REM Si una factura es seleccionada a partir de la caja de listado combo,    REM
entonces accede a ella.
    if CMDID%=701 then call cSendBtnClick ( inv1HDLG%, 206)
```

$SUB inv1Init

This subroutine is used to initialize various controls. The program INVOICE, carry the masks used to validate and format the phone field and initialize the combo box list by opening a file and reading invoice invoice numbers exist. The final bill is assigned to the key variable used to get the next available number.

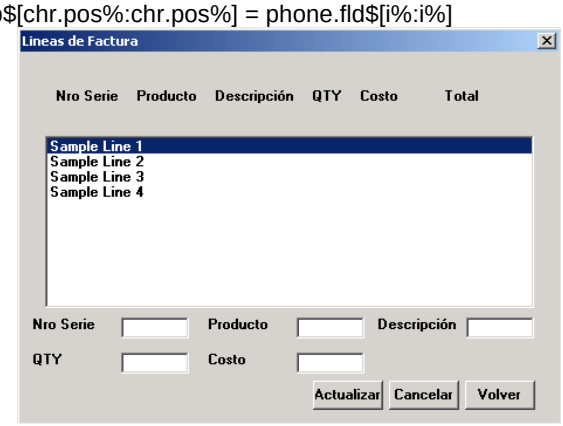
```
REM Carga las máscaras para la validación y formateo de los campos PHONE  (Teléfono).
dim phone.msk$ (6), phone.map$ (6)
for i% = 1 to 6
    read phone.msk$ (i%),phone.map$ (i%)
    next i%
data " (###) ###-####","!""#$%&' ()*+,-."
data " (   ) ###-####","!""#$%&' ()*+,-."
data "##########","""#$' ()+,-."
data "### ###-####","""#$&' ()*+,-."
data "#######","' ()+,-."
data "###-####","' ()*+,-."
REM Inicializa la caja de listado combo con los números actuales de factura.
dim invcs$ (10000)
REM Abre un fichero indexado y busca las claves  ( el número de factura).
REM Utiliza la última clave válida para asignar un número nuevo de factura.
open #1:".\invtst", update indexed
i% = 1
lastKey$ = ""
newInvoice% = 1

while 1
   readnext #1,key$: dummy$
   if eof (1) then break
   invcs$ (i%) = key$  \  i% = i% + 1  \  lastKey$ = key$
wend
if lastKey$ <> "" then newInvoice% = VAL (lastKey$) + 1

REM Inicializa el control de combo.  Se asegura que el último elemento de la REM matriz es
puesto a NULL ya que cAddComboContents leerá toda la matriz
REM hasta que se encuentre un elemento NULL.
invcs$ (i%) = ""
call cAddComboContents ( inv1HDLG%, 701, MATADDROF (invcs$) )

REM Limpia los flag de documento modificado y factura nueva y cualquier
REM matriz que no sea utilizada más adelante.
docModified% = 0
newFlag% = 0
dataEntered% = 0
clear invcs$

REM Desactiva los controles hasta que el número de factura haya sido
REM introducido.
gosub WaitForInvoice
```

The routines called with a GOSUB statement introduced in the last Access Area between REMs and usrend usrstart. are:

```
clear.ctrls:
    REM Limpia todos los controles...
    REM Primero, los controles de edición y luego las cajas combo:
    for ctrl% = 101 to 122
        CALL cSetCtrlValue ( inv1HDLG%, ctrl%, "" )
    next ctrl%
    CALL cSetCtrlValue ( inv1HDLG%, 701, "" )
    CALL cSetCtrlValue ( inv1HDLG%, 700, "" )
    return
```

```
display.ctrls:
    REM Saca por pantalla todos los valores de control del registro.
    ctrl% = 101
    for i% = 1 to 22
        CALL cSetCtrlValue ( inv1HDLG%, ctrl%, rec$ (i%) )
        ctrl% = ctrl% + 1
    next i%
    REM Da valores al control combo con las claves.
    CALL cSetCtrlValue ( inv1HDLG%, 700, rec$ (23) )
    return

WaitForInvoice:
    REM Desactiva todos los controles excepto los de invoice, Get, Search,
    REM New, Cancel y Exit. Primero los controles de edición:
    for ctrl% = 101 to 122
            CALL cEnableCtrl ( inv1HDLG%, ctrl%, 0 )
    next ctrl%
    REM Los controles combo:
    CALL cEnableCtrl ( inv1HDLG%, 700, 0 )
    REM El botón línea elementos/fichero:
    CALL cEnableCtrl ( inv1HDLG%, 201, 0 )
    CALL cEnableCtrl ( inv1HDLG%, 202, 0 )
    REM Activa Get/search/new/invoice:
    call cEnableCtrl ( inv1HDLG%, 206, 1 )
    call cEnableCtrl ( inv1HDLG%, 200, 1 )
    call cEnableCtrl ( inv1HDLG%, 205, 1 )
    call cEnableCtrl ( inv1HDLG%, 701, 1 )
    return

EnterInvoice:
    REM Activa todos los controles excepto Invoice, Get, Search, New,
    REM Cancel y Exit. Primero, los controles de edición:
            for ctrl% = 101 to 122
            CALL cEnableCtrl ( inv1HDLG%, ctrl%, 1 )
    next ctrl%
            REM Los controles combo:
            CALL cEnableCtrl ( inv1HDLG%, 700, 1 )
            REM El botón de línea elemento/fichero:
            CALL cEnableCtrl ( inv1HDLG%, 201, 1 )
            CALL cEnableCtrl ( inv1HDLG%, 202, 1 )
            REM Desconecta Get/search/new/invoice:
            call cEnableCtrl ( inv1HDLG%, 206, 0 )
    call cEnableCtrl ( inv1HDLG%, 200, 0 )
    call cEnableCtrl ( inv1HDLG%, 205, 0 )
    call cEnableCtrl ( inv1HDLG%, 701, 0 )
    return

calc.total:
    REM Calcula el campo total añadiendo los portes, impuestos, y los
    REM elementos de la línea.
    frt$ = "" \ tax$ = ""
    call cGetEditText ( inv1HDLG%, 121, addrof (frt$)) \ frt = val (frt$)
    call cGetEditText ( inv1HDLG%, 122, addrof (tax$)) \ tax = val (tax$)
    total = frt + tax
    for num% = 1 to 50
        REM Las líneas comienzan en rec$ (24) y tienen 7 campos
        REM busca el campo o accede al campo 7 :
        idx% = 24 +  ( (num% - 1) * 7) + 6
        total = total + val (rec$ (idx%))
```

```
        next num%
        total$ = str$ (total)
        return

    date.format:
        REM Da formato a los campos fecha
        date.fld$ = ""
        call cGetEditText ( inv1HDLG%, cmdid%, addrof (date.fld$))
        date.fld$ = dte$ (date.fld$)
        return

    phone.format:
        REM Da formato a los campos teléfono
        phone.fld$ = ""
        call cGetEditText ( inv1HDLG%, cmdid%, addrof (phone.fld$))
        patt$ = "" \ pmap$ = ""
        for i% = 1 to 4
          for j% = 1 to 6
              if match (phone.fld$,phone.msk$ (j%))
                patt$ = phone.msk$ (j%) \ pmap$ = phone.map$ (j%)
                j% = 6 \ i% = 4
                ifend
              next j%
          next i%
        if patt$<>""
          ofmt$ = " (510) 000-0000"
          temp$ = ofmt$
          if patt$="######" and pmap$[5:5]="%" then temp$=temp$[1:5]
              for i% = 1 to len (patt$)
                chr.pos% = asc (pmap$[i%:i%])-32
                temp$[chr.pos%:chr.pos%] = phone.fld$[i%:i%]
```




```
              next i%
              phone.fld$ = temp$
              ifend
        return
```

7.14.3 Dialog source files

When the operator selects the "Invoice Line Items" (invoice lines) in the display format, the program will make a call to the main subroutine invoice lines dialog to modify an existing invoice lines, creating lines a new bill

The source code for this dialog is stored in the file dlgline.b. The following source code has been added in several areas reserved.

The first safe area contains the following global statements that are used to pass information between these subroutines in the display format and dialogue invoice lines. These same global variables are declared in the file also dlginv.b**.**

```
REM@#      Dialog line V2.3
REM@# 1ststart
```

```
$GLOBAL mat rec$
$GLOBAL newFlag%
$GLOBAL dataEntered%
$GLOBAL docModified%
REM@# 1stend
#include "dlgline.IDH"
```

$SUB line

This secure area of the main subroutine is used to size the array must contain the 50 line items.

```
REM Dimensiona la matriz alfanumérica para la caja de listado de tal forma que REM tenga 50
elementos.
dim LineItems$ (50)
```

$SUB lineBtnClick

The following code has been added in the area reserved for managing the radio buttons Update, Cancel and Return:

```
select CMDID%
case 200
    REM Botón de actualización
    REM Actualización significa actualizar la línea actual con la información
    REM en los controles de edición.  Siempre tendremos un número valido
    REM actual de línea ya que el botón de actualización estará
    REM  deshabilitado hasta que un número de líneas haya sido
    REM seleccionado.
    LineItems$ (CurrentItem%) = fn.GetFromEdit$ ( CurrentItem% )
    REM Establece el flag de documento modificado que es una variable
    REM global de tal forma que el formato de factura pueda detectar que  REM la línea de
    factura ha sido modificada.
    docModified% = 1
    REM Actualiza el alfanumérico de la caja de listado de la línea, borrando
    REM el alfanumérico actual e insertando uno nuevo.
    call cLBoxDelString ( lineHDLG%, 300, CurrentItem% )
    call cLBoxInsString ( lineHDLG%, 300, LineItems$ (CurrentItem%), CurrentItem% )
    REM Limpia los controles de edición, desactiva los botones Update y
    REM Cancel y establece el focus de nuevo en la caja de listado.
    call cSetEditText ( lineHDLG%, 100, "" )
    call cSetEditText ( lineHDLG%, 101, "" )
    call cSetEditText ( lineHDLG%, 102, "" )
    call cSetEditText ( lineHDLG%, 106, "" )
    call cSetEditText ( lineHDLG%, 107, "" )
    call cEnableCtrl ( lineHDLG%, 200, 0 )
    call cEnableCtrl ( lineHDLG%, 201, 0 )
    call cSetFocus ( lineHDLG%, 300 )
    $exit
case 201
    REM Botón cancelar
    REM Limpia los controles de edición, desactiva los botones de Actualización
    REM y Cancelación y establece el focus de nuevo en la caja de listado.
    call cSetEditText ( lineHDLG%, 100, "" )
    call cSetEditText ( lineHDLG%, 101, "" )
    call cSetEditText ( lineHDLG%, 102, "" )
    call cSetEditText ( lineHDLG%, 106, "" )
    call cSetEditText ( lineHDLG%, 107, "" )
    call cEnableCtrl ( lineHDLG%, 200, 0 )
    call cEnableCtrl ( lineHDLG%, 201, 0 )
    call cSetFocus ( lineHDLG%, 300 )
    $exit
case 203
    REM Botón de retorno
    REM Si es un nuevo registro, activa un flag para indicar que ya se ha REM introducido
```

```
        REM un dato en la línea de pantalla.  Esto se hace
        REM para que el programa conozca que esto estaba aquí de antes
        REM en caso de que volvamos al diálogo de línea
        if newFlag%<>0 and dataEntered%=0 then dataEntered% = -1
    cend
```

$SUB lineEvent

The code that has been added in this subroutine is used to detect that an item in the list box has been selected. The code that processes the selection is in the reserved area at end of file.

```
    case 2
      REM Procesa una selección de la caja de listado para el control combo
      GOSUB select.an.item
```

$SUB lineInit

Added code to this subroutine to initialize the list box with the existing lines. The Update and Cancel buttons are disabled when there are no lines.

```
    REM Saca por pantalla las líneas en la caja de listado.  Existen 50 líneas con
    REM 7 campos que comienzan en rec$ (24).
    for item% = 1 to 50
      LineItems$ (item%) = fn.GetFromRec$ ( item% )
      next item%
    call cAddLBoxContents ( lineHDLG%, 300, MATADDROF (LineItems$) )
    REM La primera vez en el diálogo para una nueva factura, el primer elemento se
    REM selecciona automáticamente.  Los siguiente elementos deberán ser seleccionados
    REM manualmente.
    if newFlag%<>0 and dataEntered%=0
      CALL cLBoxSetSel ( lineHDLG%, 300, 1)
      CALL cLBoxTopIndex ( lineHDLG%, 300, 1)
      gosub select.an.item
    else
      REM Desactiva los botones de Actualización y Cancelación.
      call cEnableCtrl ( lineHDLG%, 200, 0 )
      call cEnableCtrl ( lineHDLG%, 201, 0 )
    ifend
```

The following code is inserted between REMs and usrend usrstart end of file:

```
    REM El formato del alfanumérico de la caja de listado es el número de elemento, el número
    REM de serie, el producto, descripción, cantidad, costo y total.  Esta función toma un
    REM elemento de la matriz del registro y lo formatea como un alfanumérico de la caja de
    REM listado toma un argumento, el número de línea
    def fn.GetFromRec$ ( num% )
      REM Los elementos de línea comienzan en rec (24) y tiene 7 campos
      idx% = 24 +  ( (num% - 1) * 7)
      fn.GetFromRec$ = RPAD$ (rec$ (idx%),4) + ~
                       RPAD$ (rec$ (idx%+1),12) + ~
                       RPAD$ (rec$ (idx%+2),10) + ~
                       RPAD$ (rec$ (idx%+3),20) + ~
                       RPAD$ (rec$ (idx%+4),6) + ~
                       FORMAT (val (rec$ (idx%+5)), "#######.##") + ~
                       FORMAT (val (rec$ (idx%+6)), "#######.##")
    fnend
    REM La función que toma una línea de los controles de edición, actualiza la matriz del
    REM registro y la formatea como una caja de listado, también toma un argumento,
    REM el número de línea.
    def fn.GetFromEdit$ ( num% )
      REM Los elementos de la línea comienzan con rec (24) y tienen 7 campos.
      idx% = 24 +  ( (num% - 1) * 7)
      REM Copia el contenido del control de edición a la matriz del registro.
      REM El número de línea  (campo 1) nunca cambia.
      call cGetEditText ( lineHDLG%, 100, ADDROF (rec$ (idx%+1)) )
```

```
    call cGetEditText ( lineHDLG%, 101, ADDROF (rec$ (idx%+2)) )
    call cGetEditText ( lineHDLG%, 102, ADDROF (rec$ (idx%+3)) )
    call cGetEditText ( lineHDLG%, 106, ADDROF (rec$ (idx%+4)) )
    call cGetEditText ( lineHDLG%, 107, ADDROF (rec$ (idx%+5)) )
    REM El campo 7  (Total) es calculado como  (Qty * Cost)
    rec$ (idx%+6) = STR$ (VAL (rec$ (idx%+4)) * VAL (rec$ (idx%+5)))
    REM Formatea un alfanumérico de la caja de listado con los resultados.
    fn.GetFromEdit$ =RPAD$ (rec$ (idx%),4) + ~
                        RPAD$ (rec$ (idx%+1),12) + ~
                        RPAD$ (rec$ (idx%+2),10) + ~
                        RPAD$ (rec$ (idx%+3),20) + ~
                        RPAD$ (rec$ (idx%+4),6) + ~
                        FORMAT (val (rec$ (idx%+5)), "#######.##") + ~
                        FORMAT (val (rec$ (idx%+6)), "#######.##")
fnend
select.an.item:
    REM Obtiene el índice del elemento seleccionado.
    call cGetLBoxSelection ( lineHDLG%, 300, MATADDROF (lineITEMS%))
    CurrentItem% = lineITEMS% (2)
    REM Corta el alfanumérico en sus campos respectivos y activa los controles de edición.
    call cSetEditText ( lineHDLG%, 100, TRIM$ (LineItems$ (CurrentItem%)[ 5:16]) )
    call cSetEditText ( lineHDLG%, 101, TRIM$ (LineItems$ (CurrentItem%)[17:26]) )
    call cSetEditText ( lineHDLG%, 102, TRIM$ (LineItems$ (CurrentItem%)[27:46]) )
    call cSetEditText ( lineHDLG%, 106, TRIM$ (LineItems$ (CurrentItem%)[47:52]) )
    call cSetEditText ( lineHDLG%, 107, TRIM$ (LineItems$ (CurrentItem%)[53:62]) )
    REM Establece el focus al primer control de edición.
    call cSetFocus ( lineHDLG%, 100 )
    REM Activa los botones de Actualización y Cancelación.
    call cEnableCtrl ( lineHDLG%, 200, 1 )
    call cEnableCtrl ( lineHDLG%, 201, 1 )
return
```

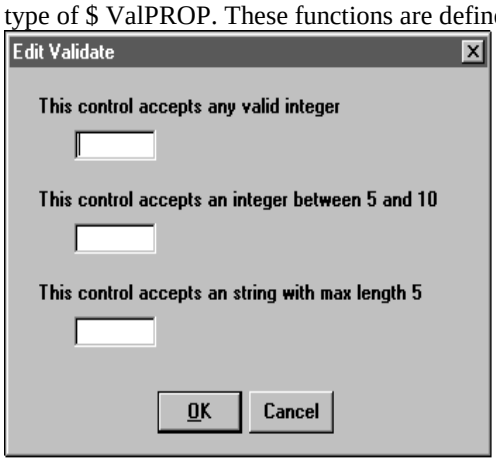
7.15VALIDATE - Data validation subroutine

It has created a simple dialogue to illustrate how data can validate user input. Three controls are used text editors to accept three inputs: an integer, an integer from a specified range, and five or fewer alphanumeric characters. The field type that can be entered is determined by the first user property defined in the control.
It uses two command buttons. The Cancel button will cancel the dialog and exit the dialog. The OK button closes the dialog, the user input display (defined as $ GLOBAL) and again calls the dialog.
The following main program code is stored in valmain.b

```
$MAIN
$GLOBAL ValENDBTN%
$GLOBAL ValEDIT100$
$GLOBAL ValEDIT101$
$GLOBAL ValEDIT102$
again:
call Val
if ValENDBTN% = 201 \  rem the Cancel button
        call cForceExit
        quit
else
        print ValEDIT100$
        print ValEDIT101$
        print ValEDIT102$
ifend
goto again
```

The VALIDATE program can be run from command line or from W32APBLD after loading the project file (validate.prj). The program will display the dialog box shown here.
The source code is stored in the dialogue file val.b. Subroutines have been modified and ValInit ValEvent. We have included the code of attachments and comments in the manual.

Note that the data type defined in the first user defined property of the controls. Also use two user roles: $ fn.Parse determines the index in the array CMDID% $ ValPROP fn.Findx% and function to read the data type of $ ValPROP. These functions are defined in the last area of security in the subroutine Valinit.


Edit Validate
This control accepts any valid integer
This control accepts an integer between 5 and 10
This control accepts an string with max length 5
OK
Cancel

```
$SUB ValEvent
SELECT CET_EVTYPE%

REM@# evtstart
case 7
    REM If it's the Cancel button or NOT an Edit control or the next
    REM control to receive the focus is 0, then ignore the Lost Focus
    REM event - otherwise validate the Edit field.
    if cet_evparm1%=201 OR  (cmdid%/100)<>1 OR cet_evparm1%=0
        $exit
    ifend
    REM Get the edit field and get and parse the control type string
    REM defined in the first user-defined property for the control.
    call cGetCtrlValue ( ValHDLG%, CMDID%, ADDROF (val$) )
    select fn.Parse$ ( ValPROPS$ ( fn.FindIdx% ( CMDID% ), 1 ) )
    case "INT"
        REM Validate as an integer.
        if val$ = ""  OR  NOT NBR (val$)
            msgbox "Please enter an integer"
            goto bad
        ifend
        $exit
    case "INTRANGE"
        REM Validate as an integer range  (min,max)
        msg$="Please enter an integer between "+min$+" and "+max$
        if val$ = ""  OR  NOT NBR (val$)
            msgbox msg$
            goto bad
        else if VAL (val$) < VAL (min$)  OR  VAL (val$) > VAL (max$)
            msgbox msg$
            goto bad
            ifend
        ifend
        $exit
    case "STRINGLEN"
        REM Validate as a string of max characters.
        msg$="Please enter a string of "+max$ + " characters or less"
        if val$ = ""  OR  LEN (val$) > VAL (max$)
            msgbox msg$
            goto bad
        ifend
        $exit
    otherwise
        $exit
    cend
bad:
REM If an entry is determined to be incorrect, focus is set back to REM the field for re-entry.
    call cSetFocus ( ValHDLG%, CMDID% )
    $exit
```

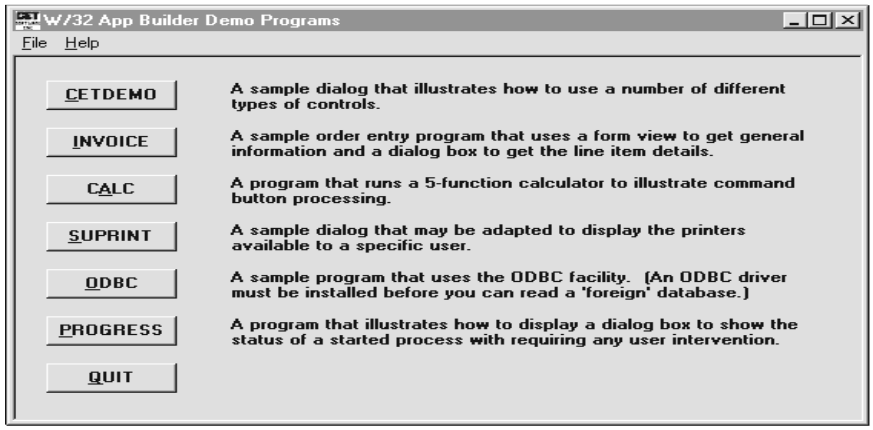
```
REM@# evtend
CEND
$EXIT

$SUB ValInit
REM     Local function/gosub save area
REM@# usrstart
REM Find the index of the control with ID=idx%
def fn.FindIdx% ( idx% )
for i% = 1 to ValCTRLS.CNT%
      if ValCTRLS% (i%) = idx%
                  fn.FindIdx% = i%
                  goto fn1end
      ifend
next i%
msgbox "Control "+STR$ (idx%)+" was not found!"
fn.FindIdx% = 0
fn1end:
fnend
REM Parse the control type string into its substrings.
REM We use a "brute force" method as all of the validation
REM occurs during user interface code where blinding speed
REM is not important.  (Remember that processors are rated in
REM millions of instructions per second.)

def fn.Parse$ ( src$  )
aa$ = src$
pos% = SCH ( 1, src$, ":" )
if pos% = 0
      type$ = src$
else
      type$ = LEFT$ ( src$, pos% - 1 )
ifend
select type$
case "INTRANGE"
      REM format is: "INTRANGE:min,max" with no embedded spaces
      pos% = SCH ( 1, src$, ":" )
      src$ = RIGHT$ ( src$, pos% + 1 )
      pos% = SCH ( 1, src$, "," )
      min$ = LEFT$ ( src$, pos% - 1 )
      max$ = RIGHT$ ( src$, pos% + 1 )
case "STRINGLEN"
      REM format is: "STRINGLEN:max" with no embedded spaces
      pos% = SCH ( 1, src$, ":" )
      max$ = RIGHT$ ( src$, pos% + 1 )
case "INT"
      REM nothing else to do
otherwise
      msgbox "Unknown control type: " + src$
      type$ = "UNKNOWN"
cend
fn.Parse$ = type$
fnend

REM@# usrend
```

7.16W32 DEMOS - A menu example program

The program W32DEMOS we made it easier to run programs w/32 examples. This program also shows how easy it is to create a menu using buttons. Each button has a corresponding label that describes the purpose of the command.
When running the following menu appears:

The main program code (which contains the sentence $ MAIN) is stored in the file as follows w32demos.b. Since this file does not need any additional information, do not see any sentence $ GLOBAL:

.


W/32 App Builder Demo Programs
File Help
CETDEMO
A sample dialog that illustrates how to use a number of different types of controls.
INVOICE
A sample order entry program that uses a form view to get general information and a dialog box to get the line item details.
CALC
A program that runs a 5-function calculator to illustrate command button processing.
SUPRINT
A sample dialog that may be adapted to display the printers available to a specific user.
ODBC
A sample program that uses the ODBC facility. (An ODBC driver must be installed before you can read a 'foreign' database.)
PROGRESS
A program that illustrates how to display a dialog box to show the status of a started process with requiring any user intervention.
QUIT

```
$main
call cAppTitle ("W/32 App
Builder Demo Programs")
call cForceExit
call w32dlg
quit
```

The subroutine w32dlgBtnClick has been modified to process the selection of the command buttons. Note that each time you press a button with the mouse to run one of the example programs, making a call to a function that minimizes the main application window before the start of the process chosen. To start the example that has been chosen cCreateProcess function is used, this function can determine when the user leaves the program example (and terminates the process) in order to restore the main window (which was minimized). This process is performed in the subroutine Status:

```
$SUB w32dlgBtnClick
REM     Button Click save area
REM@# btnstart
select cmdid%
case 201
        call cAppWindowMin
        call cCreateProcess (addrof (hdle%),".\cetdemo",3)
        gosub status
case 202
        call cAppWindowMin
        call cCreateProcess (addrof (hdle%),".\invoices",3)
        gosub status
case 203
        call cAppWindowMin
        call cCreateProcess (addrof (hdle%),".\calc",3)
        gosub status
case 204
        call cAppWindowMin
        call cCreateProcess (addrof (hdle%),".\suprint",3)
        gosub status
case 205
        call cAppWindowMin
        call cCreateProcess (addrof (hdle%),".\odbc",3)
        gosub status
case 206
        call cAppWindowMin
        call cCreateProcess (addrof (hdle%),".\progress",3)
        gosub status
        cend
REM@# btnend
GOSUB w32dlg.BTNEND
$EXIT
```

The Status subroutine is stored in the secure area of the last file in the subroutine w32dlgInit. See Chapter w/32 Window Functions for more information on the use of functions and cProcessStat cCreateProcess.

```
$SUB w32dlgInit
REM     Local function/gosub save area
REM@# usrstart
status:
        if hdle%=0
                msgbox "Program not started"
                call cAppWindowNormal
        ifend
        stat%=998
        while stat%=998
                sleep .25
                call cProcessStat (hdle%,addrof (stat%))
        wend
        call cAppWindowNormal
return
REM@# usrend
```

7.17 CALC - A special buttons proccesing program

This program example shows a 5-function calculator. Each key on the calculator is a command button. For this program we have made some changes in the file dialog.
At the end of the subroutine is calcBtnClick cSetCtrlValue a call to update the text editor (The calculator screen) as normal calculators work.
Added code in the security area calcInit subroutine to position the cursor at the end of the line of text editor (The starting point of the calculator screen). Tl and as expected, the data will be right justified.
The main program code is very simple:
$ main
call calc
call cForceExit
quit
The source code file calc.b dialog has been modified and is also included in the manual. Note that the ON ERROR GOTO statement at the beginning of the subroutine calcBtnClick. Since each BASIC subroutine can have its own error handler, this code should be included in each subroutine that could cause an error. (We only have included it in this example). The error handling routine is stored in the security area of the end of the program.

```
$SUB calcBtnClick
REM     Button Click save area
REM@# btnstart
on error goto calcerr
call cGetCtrlValue ( calcHDLG%, 100, ADDROF (value$) )
value = VAL (value$)
key% = CMDID% - 200
select
case key% <= 9
        REM a numeric key
        if state$ = "Clear" OR state$ = "Result1"
                disp$ = str$ (key%)
                state$ = "Append"
        else if state$ = "Append"
                disp$ = value$ + str$ (key%)
        ifend
        ifend
case key% = 10
        REM a decimal point
        if state$ = "Clear" OR state$ = "Result1"
                disp$ = "0."
                state$ = "Append"
        else if state$ = "Append"
                disp$ = value$ + "."
        ifend
        ifend
case key% = 11
        REM the change sign key
        value = 0 - value
        disp$ = str$ (value)
case key% = 12
        REM the equal key
        select cmd%
        case 13
                result = num1 + value
        case 14
                result = num1 - value
        case 15
                result = num1 * value
        case 16
                result = num1 / value
        cend
        disp$ = str$ (result)
        state$ = "Clear"
        cmd% = 0
case key% <= 16
```

```
                REM a command key
                num1 = value
                state$ = "Result1"
                disp$ = value$
                cmd% = key%
        case key% = 17
                REM the Clear Entry key
                disp$ = ""
        case key% = 18
                REM the Clear all key
                disp$ = "0."  \  state$ = "Clear"
        otherwise
                MsgBox "Calc: unknown button"
        cend
        REM Update the display each time a button is pressed.
        call cSetCtrlValue ( calcHDLG%, 100, disp$ )
        REM@# btnend
        GOSUB calc.BTNEND
        $EXIT
```

The following changes have been made to initially position the cursor at the end of the text editor. It shows the initial value '0 'right justified as it would in a real calculator. The error routine is stored in the last area of security.

```
        $SUB calcInit
        REM     Initialize dialog save area
        REM@# inistart
        state$ = "Clear"
        REM set the focus to the edit control and remove the selection
        call cSetFocus ( calcHDLG%, 100 )
        call cSetCaret ( calcHDLG%, 100, -1 )
        REM@# iniend
        $EXIT
        REM@# usrstart
        calcerr:
        MsgBox "Calc: Error err="+str$ (err)+" line="+str$ (erl)
        quit
        REM@# usrend
```

7.18 PROGRESS - A dialog example with progress bar

The PROGRESS program illustrates how a push of a button you can send to activate a dialogue without any user interaction. At that time, the dialog displays a progress bar process small black bars, which increase in number as a symbol of the passage of time while the process is active, or until the user clicks the cancel button (Cancel) to close dialogue (without canceling the process started).

The main program source code is stored in the file as follows prgmain.b:

```
        $MAIN
        call prgmon
        msgbox "done"
        call cForceExit
        quit
```

The source file subroutine PrgmonBtnClick prgmon.b dialogue has been modified to detect the selection of a special button ID No. 299 (prgmonInit sent from the subroutine). If detected, it is a call to start the program cCreateProcess w/32 CALC. While CALC is active, it will show the progress bar. Note that the character CHR $ (135) (a small black grating) is used to simulate a progress bar, but you could also use any printable character.

Note that a call is made to cEndDlg to end the dialog automatically. This is the operation performed when activated End Dialog property of a command button. Because we do not alter the normal operation cancel, use the function to replace cEndDlg.

```
        $SUB prgmonBtnClick
        REM@# btnstart
        REM Check for our "special" button ID.
        if CMDID% = 299
                REM You can also start the new process in your $MAIN
                REM and make hndl% a global variable.
                call cCreateProcess ( ADDROF (hndl%), "calc.exe", 1 )
                progress$ = CHR$ (135)  \  stat% = 998
```

```
            while stat% = 998
                call cSetCtrlValue ( prgmonHDLG%, 100, progress$ )
                if LEN (progress$)<40 then progress=progress$+CHR$ (135) else
progress$=CHR$ (135)
                sleep .5
                call cProcessStat ( hndl%, ADDROF (stat%) )
            wend
            REM We are done so end the dialog…
            call cEndDlg ( prgmonHDLG%, CMDID% )
ifend
REM@# btnend
GOSUB prgmon.BTNEND
$EXIT
```

The button press is sent from the security area prgmonInit subroutine. We chose the identifier value 299, which is within the valid range of values of ID, but not worth a Real ID.

```
$SUB prgmonInit
REM@# inistart
REM Send an initial button click to start the monitor, a value in the
REM button ID range  (200-299) that is NOT the ID of a real button.
call cSendBtnClick ( prgmonHDLG%, 299 )
REM@# iniend
$EXIT
```

Chapter 8 W/32 Window Functions

8.1 Introduction

The Application Development System w/32 CET provides a wide variety of C functions that let you run operations that would be difficult or impossible to program using BASIC only. This chapter covers the functions you can use to deploy Windows standard features within their programs.
Information about special functions available for use within forms and dialog boxes are presented in chapter Dialogues Editor w/32. Special functions file, to get information from the environment are covered in Chapter routines in C.
W/32 functions are described in alphabetical order so you can locate them quickly. A quick reference table at the end of this chapter gives an overview of the features available. Remember, in the names of CALL routines are important case-sensitive. You have to enter them as shown in this manual.

8.2 Using C language routines

The CALL statement is used to transfer control to an external routine in C, BASIC or assembler. The name of the routines should be as shown in this manual (case sensitive) to the linker (LINK) to resolve references during compilation.
There are two modes of CALL statements. Mode 1 uses the syntax "CALL routine" to transfer control to a specific routine. No arguments are passed to the routine, and returns information to the main program. A few routines as cFontValues w/32 use this syntax.
Most C routines use mode 2 of the CALL statement to pass one or more arguments to the routine using the syntax "CALL routine (arg-list)". When used this way, it is important to spend the type and number of arguments in the proper order. Otherwise it will generate a runtime error.
For example, if a routine as cDIBWindow expects an integer value (handle%) has to pass an integer variable or constant, not a float value.
The special function ADDROF be used to pass the memory address of a variable to a C routine, passing the value of this variable. This variable can be an element of a table or a simple variable. (MATADDROF is a similar function to pass the memory address of a table.)
The routine will evaluate the direction of the variable and assign a new value. For example, the ADDROF (address) of two integer variables are passed cGetCursor:
CALL cGetCursor (ADDROF (row%), ADDROF (column%))
The values assigned to the variables row% and column% will be returned to the BASIC program. To say that the function values assigned as a result of these two variables, which later will be used by the BASIC program, means that the routine must know where to load the defaults so you need to know the address of memory, so it is essential to pass these two variables by ADDROF.
For more information on the CALL statement, see the Language Reference Manual CET BASIC

cAppTitle

The programs show CET w/32 w/32 App Builder along with the current version number as the default title of the main window. This function can be used to edit and display a different text as the name of your application. The CALL statement syntax is:
cAppTitle ($ title)
Where:
title
specifies the text to be used to replace the default title.
Be careful that the text does not exceed the length of the window because it exceeds any character will be truncated.
Note: This function can also be called from the subroutine cetPreInit in the resource file cetwin.b file copy, or from $ MAIN program, or from the Init subroutine, when you want to put the same title to all its programs. See chapter Windows Manager for information on how to do this.

cAppWindowInfo

This routine can be used to obtain the location and size of the main application window. The CALL statement syntax is:
cAppWindowInfo call (addrof (x%), addrof (y%), addrof (w%) addrof (h%))
Where:
x, and returns the upper left column and row of the window.

w, h return the width and height of the window.
See cSizeAppWindow function if you need to resize window loa.

cAppWindowMax

This routine can be used to maximize the application's main window after making a minimization of the window (calling cAppWindowMin). No no argument is passed to the routine. The syntax is:
call cAppWindowMax

cAppWindowMin

This routine can be used to minimize the main window of the application will not pass any arguments to the routine. The syntax is:
call cAppWindowMin

cAppWindowNormal

This routine can be used to restore the main application window to its normal size and position. (For example if you have previously done a maximization or minimization). No no argument is passed to the routine. The syntax is:
call cAppWindowNormal

cAppWindowTop

This routine can be used to carry the application window to the top of the window stack. No arguments happen. The syntax of the CALL statement:
cAppWindowTop

cCbCanPaste

This function can be used to check if the Clipboard has some content.
The syntax is:
cCbCanPaste call (addrof (rc%))
where:
rc%
This variable is used to enter a return code. This code will be:
0 when the Clipboard is empty
1 when the Clipboard has any content that can 'paste'.

cCbCopy

This function is used to copy an alphanumeric string to Cliboard. The function syntax is:
cCbCopy call (addrof (rc%), sting $)
where
string $
Is the alphanumeric string to copy
rc%
This variable is used to enter a return code. This code will be:
0 when the operation is successful
1 when the Clipboard is not able to open (probably being used by another user)

cCbEmpty

This function is used to copy an alphanumeric string to Cliboard. The function syntax is:
cCbEmpty call (addrof (rc%))
where
rc%
This variable is used to enter a return code. This code will be:
0 when the operation is successful
1 when the Clipboard is not able to open (probably being used by another user)

cCbPaste

This function is used to paste an alphanumeric string from Cliboard. The function syntax is:
cCbPaste call (addrof (rc%), sting $)
where
string $
Is the alphanumeric string to copy
rc%
This variable is used to enter a return code. This code will be:
0 when the operation is successful
1 when the Clipboard is not able to open (probably being used by another user)
2 Indicates that the Clipboard is empty.

cCheckItem

This routine is used to change the value of a radio button that indicates its current state. For example, if a menu item is mutually exclusive options such as on / off or enabled / disabled, this feature can be used to find out whether or not that option button activated. The CALL statement syntax is:
cCheckItem (item% status%)
Where:
item
ID is the value (an integer between 1 and 250) associated with the menu item. For example the value for ID_WINSUB1 ID is 1 and 250 for ID_WINSUB250
state
is an integer value, or 1 to enable or 0 to disable.
For example:

```
REM Enable radio button menu item associated
REM ID_WINSUB4
CALL cCheckItem (4, 1)
```

cChoiceList

This routine displays a search dialog in a list and allows the user to select an item (and press the OK button) or double-click on an item ..
The items are sorted automatically before displaying the window. (Use the cChoiceListNS whether to display an unordered list.) Although the list can have up to 15000 elements, in practice the limit is on a 2000 (depending on the speed of your computer). The CALL statement syntax is:
cChoiceList call ($ title, mataddrof ($ elements), addrof ($ selection))
Where:
title
specifies the string to be displayed as the window title.
elements
is the string table to be displayed in the list of options. Note that the function displays a list of items, as many as the maximum size of the table sized chains or until it is an element of the empty table.
selection
chain is selected from the list of options. A null string is returned if the operator selects Cancel.
For example, to enter a number of elements and create a list box:

```
INPUT j% \ DIM $ element (j%)
FOR i% = 1 TO j%
element $ (i%) = "line" + STR $ (i%)
NEXT I%
$ title = STR $ (j%) + "List Items"
CALL cChoiceList ($ title, MATADDROF ($ element), ADDROF ($ selection))
PRINT "You selected:" $ selection
CLEAR $ element
```

cChoiceListNS

This routine operates similarly to cChoiceList and displays a list box with displacements. The only difference is that this routine will not sort the items before they are displayed. The CALL statement syntax is:
For greater flexibility and speed, we recommend that you use this routine instead of cChoiceList to display a list of items that has been ordered by the program.
cChoiceListNS call ($ title, mataddrof ($ elements), addrof ($ selection))
Please consult cChoiceList to learn about their use.
cCloseDIBWindow
This routine is used to close a window DIB, used to display an image. The CALL statement syntax is:

cCloseDIBWindow call (handle%)
Where:
handle
is the descriptor of the window open DIB by calling cOpenDIBWindow.
Note that attempting to close a window using a descriptor invalid DIB generate an error message. See cOpenDIBWindow function for information on using Windows DIB to display graphic files in bitmap format of Microsoft Windows.

cCreateProcess

This function can be used to start a process. Under Windows NT and Windows 95, you can begin any type of process (eg DOS process, a Windows 16-bit, or 32-bit Windows application). Under Windows, you can only

use cCreateProcess starting a 32-bit Windows application, otherwise the return value indicates that the process did not start, when I really did.
The statement syntax is:
CALL cCreateProcess (ADDROF (handle%), $ cmd, show%)
where:
handle
Control is the number returned by the process. A value of zero means that the process did not start.
cmd
Is the name of the executable including the necessary arguments. The search sequence is as below:
 The directory where the calling program is executed.
 The current working directory
 The \ Windows or \ Windows \ system
 The directories set in the PATH
 directories within the variable set B_LKPATH
Normally this argument affects only the executable. Windows programs usually only look in the current working directory unless you specify the full name with path.
show
Indicates the size of the window, a window 1 is normal, 2 is a minimized window, and 3 is a maximized window or full screen window.
This argument does not affect when you start a DOS application, in this case we will use a full screen, unless you have defined a. Pif windows properly.
See cProcessStat function to see how to ask the status of a process after it has begun.
See also the program w32DEMOS for examples of using this feature.
cDIBInfo
This routine is used to get the height and width of a graph DIB. The CALL statement syntax is:
cDIBInfo call ($ name, ADDROF (a%), ADDROF (h%))
Where
name
specifies the full name including the PATH DIB file to read. The graphic file needs to be in bitmap format of Microsoft Windows.
to
returns the width of the graph DIB.
h
returns the height of the chart DIB.

cDIBWindowInfo

This routine is used to obtain the location and size of a window DIB. The syntax for the CALL statement is:

cDIBWindowInfo call (hdle% addrof (x%), addrof (y%), addrof (w%) addrof (h%))
cDIBWindowInfo call (handle% addrof (x%), addrof (y%), addrof (at%), addrof (h%))
where:
handle
refers to the window descriptor DIB information he wants.
x, y
returns the position of the upper left column and row of the window.
a, h
returns the width and height of the window.

cDIBWindowTop

This routine is used to set the specified window to the beginning DIB of the window stack. The CALL statement syntax is:
cDIBWindowTop call (handle%)
Where:
handle
descriptor specifies the DIB window to be displayed as the first window of the stack.
See cOpenDIBWindow function to find out how the windows are used to display DIB graphic files in bitmap format of Microsoft Windows.

cDisplayInfo

This routine is used to obtain the resolution and number of colors in the display. The CALL statement syntax is:
cDisplayInfo call (addrof (x%), addrof (y%), addrof (colors%))

Where:
x, y
is the horizontal and vertical resolution (eg 640x480) that are using the device.
colors
is the number of colors available. This value is returned as bits per spot. The value 4 indicates that they have 16 colors and 8 there are 256. A value of 1 indicates that the monitor is monochrome.
Note that the value returned informing the driver's color depends on the particular video adapter being used. Report that sometimes we have found that a color monitor returns the value 1.

cEnableDocking

You can use this routine to enable a movable toolbar. When a toolbar is 'movable', the operator can drag it out of the application window, so that it is displayed in its own window. The statement syntax is:
call cEnableDocking
When the coupling is enabled, there will be a black vertical line to the right of the last button. To hide the toolbar to move the mouse in the background area that is next to the buttons and the black line. To reconnect the toolbar to return to the beginning or end drag and drop. (The cutting line display will change to a solid line when above a cleavage site).

cEnableItem

CEnableItem routine can be used to enable or disable a menu item ID_WINSUB. This feature is used especially when you want to ensure that the operator can not select an item that could disrupt a particular operation and cause problems. The statement syntax is:
cEnableItem call (element% status%)
Where:
element
ID is the value (an integer between 1 and 99) associated with the menu item. For example, the ID value should be 1 to 99 for ID_WINSUB99 ID_WINSUB1.
state
is an integer value, or 1 to enable or 0 to disable.
For example:
REM Disables the menu item associated ID_WINSUB4 won.
CALL cEnableIterm (4, 0)

By default, all menu items are enabled so that they can be selected. Disabling a menu item text changes to gray and can not be selected.
See chapter The Windows Framework for information on how to modify the default menu and / or create new menu items.
cEnableMainWin
CEnableMainWin routine can be used to control the activation status of the main application window. This feature is useful if you plan to move the window and make it invisible until it is in the final position (with calls cShowMainWin). During this process, you may need to disable any mouse or keyboard input. The syntax of a sentence is:
cEnableMainWin call (status%)
state
is an integer, 1 to enable the main window (mouse input and keyboard), or 0 to disable the window.

cEnumPrinters

This routine is used to obtain information from the available printers. Typically, this feature is used under a BASIC program when you need to get information from the printer to activate B_PRINTERn variable with the specific name of the printer making a function call Bputenv. (Refer to implementation notes in the appendix for more information on printing).
The syntax of the call is:
cEnumPrinters call ($ state, addrof (prtname $), addrof (drvname $), addrof ($ port))
Where:
state
It is the state, 0 begin the process of specifying, 1 will continue, and 2 free the dynamic memory associated with the enumeration process.
prtname
Is used to return the text "printer name" to be used to activate B_PRINTERn. Under Windows NT and 95, this is the printer name. Under Windows is the name of the driver.
drvname

This is the variable used to retrieve alphanumeric "Driver Name" under NT 95 or the name of the driver file using Windows. In the latter case, the file name is returned for information only. It is not necessary to print handler.

port

It is the alphanumeric variable used to return the port to which the printer is connected. Under Windows NT and 95, the port number is important.

Note that the last three parameters will be empty when all printers are specified.

Currently, cEnumPrinters function can be used to obtain information of the printer recognized by the printer driver. Although they may have more available printers as network printers that are not connected to the printer driver, Windows does not provide any means to determine if a printer is local or networked. Under NT and 95, the name of a network printer normally begins with "\ \" as a path name to the printer, so it will be possible to implement this feature in future releases.

The following piece of program illustrates how to use this feature. One example of W32usa programs this feature. The project also illustrates this feature suprint.prj.

```
REM begins to list the printers
cEnumPrinters call (0, addrof ($ prt), addrof ($ drvname, addrof ($ port))
while prtname $ <> ""
print $ prtname, drvname $, $ port
cEnumPrinters call (1 addrof ($ prt), addrof (drvname $), addrof ($ port))
wend
REM releases dynamic memory being used
call cEnumPrinters (2 addrof ($ prt), addrof (drvname $), addrof ($ port))
```

cFontValues

This routine can be used to get the values to pass cSelectFont to change the current font. It requires no arguments. The CALL statement syntax is:

```
call cFontValues
```

For example:

```
PRINT "Any text or line graph to show"
LINPUT to $
CALL cFontValues
END
```

Run the program once compiled and select the desired font from the Font menu. All values requiring routine cSelectFont be shown by a dialog box. This procedure was discussed in detail in section Changing defaults Framework Windows in the previous chapter.

cForceExit

This routine is used to force the program to end without displaying the OK button in the message box. Takes no arguments. The CALL statement syntax is:

```
call cForceExit
```

When it reaches the END statement, the default action is to display a message box and wait for the operator to click on OK. If you call cForceExit before running an END statement will cause the program to exit immediately (and close the window).

cGetCursor

This function is used to get the current position of the cursor column and row. The return values are relative to the current window. The CALL statement syntax is:

```
cGetCursor call (ADDROF (row%), ADDROF (column%))
```

Where:

row

returns the row or coordinate.

column

returns the column or coordinate x.

cGetMouse

This function is used to return the mouse position in screen coordinates are relative to the window. The statement syntax is:

```
cGetMouse call (ADDROF (row%), ADDROF (column%))
```

Where:

row

returns the current row or coordinate.

column

returns the current column or coordinate x.

cInputQ
This function is used to enter a key code into the input queue or buffer exactly as if the key had been entered by the operator. The statement syntax is:
cInputQ call (key%)
Where:
key
is the ASCII value of the key to be inserted in the input buffer.
For example:
REM introduces an ESC within the input queue
CALL cInputQ (27)
REM introduces the F1 key in the input queue
CInputQ CALL (0)
CALL cInputQ (59)
To add a special key, such as F1, must enter two key codes within the buffer. (Eg a zero and extended ASCII code for the F1 key). A key that does not correspond to a printable character (eg function keys, cursor keys, insert, home, etc ...) is considered as special key.
Extended ASCII values are the same as the sentence BASIC GET returns. See Appendix Manual CET BASIC Language Reference for using the values of the character table.

cLogin

This routine is used to display a login dialog box the operator to enter a user name and a password. The return values that can be used within your application. The CALL statement syntax is:
cLogin call (ADDROF ($ user), ADDROF ($ key))
Where:
user
is a specific user name entered by the user.
key
is the password of the user.
For safety purposes, the asterisks show cLogin key field even when entering characters.
This feature is especially relevant when the program should create temporary work files in a multi-user environment. The following example shows how to create an indexed file with a unique name for each user. The file will be archived in the \ tmp Z drive on a remote server.
cLogin call (ADDROF ($ user), ADDROF ($ password))
REM The string will be filled depending on the content
REM username
$ filename = "Z: \ TMP \ WORK" & user $
Berase call ($ filename)
Bcreate call ("filename $ &" INDEXED keylen RECLEN 100 10 ")
OPEN # 1: $ filename, update indexed
This procedure will work equally with sequential files. In this case, instead of the functions and Berase Bcreate, use OPEN with the option to write the file OUTPUT, or EXTEND option to add new data to the end of file.

cMaxSize

This routine is used to size a window to the maximum supported size, 25 lines of text using the current font. If the window is already full size, this function has no effect. It requires no arguments. The syntax is:
call cMaxSize
cMessageBoxOkC
CMessageBoxOkC function is provided to display a message box with OK and Cancel buttons. The program will be suspended until the user selects a button.
cMessageBoxOkC requires two arguments: a string to be displayed in the message box and an integer variable that will be used to return the result.
The statement syntax is:
CALL cMessageBoxOkC ($ menu, ADDROF (result%))
Where:
men
specifies the string to display.
result
is a value indicating which button was selected. The return value is 1 for OK, 0 for Cancel.
For example:
CALL cMessageBoxOkC ("Print Listing", ADDROF (result%))
IF result%

Then MsgBox "Printing Document"
ELSE
MsgBox "The impression has been canceled"
IfEnd
Note: MSGBOX judgment has been used to display a message box with only OK button .. This statement is covered in the Implementation Notes section of the Appendix.

cMoveAppWindow

This routine is used to move the application window to a new location. The CALL statement syntax is:
cMoveAppWindow call (x%, y%)
Where:
x, y
specifies the positions of the new upper left column and row to the window.
For example:
REM moves the upper left corner of the application window to
REM display the location of 50, 50
CALL cMoveAppWindow (50, 50)

cMoveDIBWindow

This routine can be used to move a window to a new location DIB. The CALL statement syntax is:
cMoveDIBWindow call (handle%, x%, y%)
Where:
handle
is the descriptor of the window returned by the call to cOpenDIBWindow. Trying to move a window with an invalid handle generate an error.
x, y
are the coordinates of the upper left corner of the new position of the window.
For example:
REM Move the upper left corner of the window to new
REM screen location 50, 50
CALL cMoveDIBWindow (dibwin1%, 50, 50)
See cOpenDIBWindow function to obtain the necessary information on how to manage the windows DIB to display graphic images in bitmap format of Microsoft Windows.

cNewMenu

This routine is used to display a new menu. The syntax is:
cNewMenu call (menu%)
Where:
menu
is an integer from 1 to 50 that specifies the menu ID. For example, the ID value is 2 for IDR_MENU2, 50 for IDR_MENU50 and 1 for the default menu: IDR_MAINFRAME.
For example:
REM Load a new menu associated with IDR_MENU5.
CALL cNewMenu (5)
REM returns to the initial menu associated with IDR_MAINFRAME
CALL cNewMenu (1)
When you start your application, the initial menu displayed is the one associated with the identifier IDR_MAINFRAME. If you require multiple menus, you can define them in the. Rc using predefined identifiers to IDR_MENU50 IDR_MENU2 go.
cNewMenu be called to load a new menu at any time during program execution. To return to the initial menu, make another call by, menu% = 1.
See chapter Framework Windows for details about the use of menus in a Windows application.

cOnWin32s

COnWin32s routine can be used to determine whether the current system is rolling under Win32s (Windows 3.1 or Windows for Workgroups). The syntax is:
cOnWin32s call (ADDROF (flag%))
Where:
flag
has a value of one if you are shooting under Windows or Windows for Workgroups, or zero if it is rolling under Windows NT or 95.

cOpenBrowser

This routine can be used to display a dialog box (browser) open file and return the program to the selection made by the user. The statement syntax is:
cOpenBrowser call (ADDROF ($ name), $ filter)
Where:
name
is the string used to return the name of the selected file.
filter
a chain is used to restrict the files displayed in the list.
For example:
REM Shows a dialog box for selecting files to open
REM and restricts basic sources files (*. B)
CALL cOpenBrowser (ADDROF ($ name), "Basic Files (*. B) | *. B | |")
This function displays a standard Windows command window open in File menu. If a file is selected will return the full path name (including the unit). If the user chooses the Cancel button. $ Name is loaded with zero. When $ filter is an empty string, the files appear in the list, as if it had been specified *. *
The filter string must be formatted with two entries separated by a vertical bar ('|') and end with two or more vertical bars. The first entry is an informational message about the description of the filter. The second entry should indicate the file extension to be used to some extent as a restriction of the list of files in a manner similar to the DOS DIR command. In the second inning may have multiple extensions separated by a semicolon (';') as shown below:
"Source files (*. B, *. Asm) | *. B, *. Asm | |"
It may also include multiple filters in which each filter can be accessed via a drop-down window. For example:
"Basic Files (*. B) | *. B | Assembler Files (*. Asm) | *. Asm | |"
The home directory is displayed in the browser is the current working directory. If you use a different directory, change the directory with a call to the function before calling Bchdir cOpenBrowser. To search the current working directory to return to it once you've returned from the browser function makes a call to Bgetcwd. For example:
REM Save the current working directory
Bgetcwd call (addrof (CURDIR $))
REM Change to the directory from which you want to start the browser
Bchdir call ("C: \ MyApp \ subdir")
REM Set the default file name
$ name = "default.txt"
cOpenBrowser call (addrof ($ name), "")
REM Return to directory
Bchdir call (CURDIR $)

cOpenDIBWindow

This routine can be used to open a new window that displays a bitmap file Microsoft Windows, ie any. Dib or. Bmp. This is a powerful feature because it allows you to read graphics files and display a graph or chart without having to leave your program. For example, if you have an insurance package may have a button in the toolbar, you can choose to display the picture of the insured.
When a call is made to cOpenDIBWindow, the new window is inactive and the user has to click with the mouse to see it. (Your program can execute a call to cDIBWindowTop to move the window to the top (active) without user intervention.)

The bitmap graphics will be resized (enlarged or reduced) to fit within the window. If the width and height arguments are zero, it will create a window the same size as the bitmap. The CALL statement syntax is:
cOpenDIBWindow call (ADDROF (handle%), $ path, $ title, x%, y%, a%, h%)
Where:
handle
is the descriptor of the DIB window. This descriptor should be used to reference the window within other function calls DIB (p.ej.cDIBWindowsTop)
path
is the full path name (location) of the file. dib or. bmp to show.
title
is the string to be displayed as the window title.
x, y
are the column and row upper left of the new window.
a, h

are the width and height of the window. If these values are zero, the window will be the same size as the bitmap.
For example:
REM opens the file \ tmp \ foto.bmp in a 256x256 window size
REM in locating 0.0 screen with the title "Graphic Window 1"
CALL cOpenDIBWindow (ADDROF (dibwin%), "\ tmp \ foto.bmp"
"Graphic Window 1", 0, 0, 256, 256)
The return value as a descriptor of the window must be used to reference this window into the other functions that operate on Windows DIB.
Note: Windows DIB not explicitly closed by the application will be closed on exit. They will not open or when you make a LINK CHAIN.

cPassword

This function is useful when you want to protect specific parts of your application. The statement syntax is:
cPassword call (addrof ($ pwd))

cProcessStat

This routine can be used to find the status of a process that has been started with a call to cCreateProcess. The syntax of this statement is:
CALL cProcessStat (handle% ADDROF (stat%))
where:
handle
It is the value returned by cCreateProcess control.
stat
Is a value indicating the status of the process. Any value other than 998 or 999 indicates that the process is over. A value of 998 or 999 means:
998 = The process is still active
999 = The status can not be obtained because the process is not started or not a 32-bit (under Windows)

Note that an application w/32 end, by default, with the exit code 0. You can get the exit status using the quit statement with a numeric expression in the range 0-98. Except errors that follow, values greater than 98 indicate that an error has occurred.

100 = Process ended with a judgment CHAIN / LINK / RUN
101 = program name passed CHAIN / LINK / RUN could not be found
If your program needs to wait to complete a process, be sure to perform a SLEEP or make a call to cCheckMessage otherwise Windows will appear to "hang". (Under Windows the whole system may appear to hang). For example:

```
REM Create a process to be executed on a minimized and gets its return code REM
cCreateProcess call (addrof (hdl%), "\ ats \ glpost.exe", 1)
if hdl% = 0
print "process not started"
goto NoStatus
IfEnd
stat% = 998
while stat% = 998
SLEEP 5 \ REM or call with any argument cCheckMessage
cProcessStat call (hdl% addrof (stat%))
wend
print "Process Exit Code:" stat%
NoStatus: ...
```

If your program needs to wait to complete a process, perform a SLEEP. Otherwise, the window look gets "hung". (Under Windows the complete look that hangs., For example:

```
REM Create a process for ejececutarse so get your REM minimized and return code
cCreateProcess call (addrof (hdl%), "\ ats \ glpost.exe", 1)
if hdl% = 0
print "process not started"
goto NoStatus
IfEnd
stat% = 998
while stat% = 998
SLEEP 25
```

REM Sets a value unused processing time, Rem only need to check the process.
cProcessStat call (hdl% addrof (stat%))
wend
print "process exit code:" stat%
NoStatus: ...

See examples and menu program PROGRESS, the example shows a bar set, as an example of using this feature.

cSaveBrowser

This routine is used to display a dialog box like "Save As" and return the selected item to the program that called it. The syntax is:
cSaveBrowser call (ADDROF ($ name), $ ext, $ filter)
Where:
name
is the string used to return the full path name of the selected file.
ext
is the default extension that will be added to the file name. Specify an empty string if you want to use a.
filter
is the filter used to limit the files list. An empty string will show all files.
For example:
REM Shows a dialog box like "Save As" and if the user does not
REM supplied with a filename extension used by default. B
REM No files are limited to show
CALL cSaveBrowser (ADDROF ($ name), "b", "")
The only difference between this function and cOpenBrowser is that this browser using "Save as" (save as "for the title and an optional parameter, the default extension to append to the file name if the user does not select another extension.

cSelectFont

This function can be called to change the default font used to display text. The CALL statement syntax is:
cSelectFont call ($ name, height%, width%, italic% charset% family%)
Where:
name
is the name of the font to use.
height
indicates the height or size of the source
width
indicates the width of the font or style
italic
option specifies the italic
charset
specifies the characteristics of the source.
family
specifies the font family.
To find out which parameters are used with a particular source, see the description of cFontValues. For example:
Select the source REM Terminal 9 points in the regular style.
CALL cSelectFont ("Terminal", -12, 400, 0, 255, 49)

cSetCursor

This function can be used to select the position of the cursor where you want to display the next screen output. The CALL statement syntax is:
cSetCursor call (row%, column%)
Where:
row
specifying the new location of the cursor line.
column
specifies the new column position of the cursor.

cShowMainWin

CShowMainWin routine can be used to control the visibility of the main application window. This feature is useful when you plan to move da window and wants it to remain invisible until it is placed in the new

position. (With another call to cShowMainWin). During this process, you would need to disable any mouse or keyboard input by calling cEnableMainWin.
The statement syntax is:
cShowMainWin call (state%)
Where
state
is an integer value, either the value 1 to make the main window visible or set to 0 to make it invisible.
Note that when you invoke a format (with a call to the subroutine CALL), the format will always be visible. A call to cShowMainWin (0) in the last line of dialogue routine (after a call to cShowForm and before $ EXIT) lets you go from one format to another without displaying a text-based window during the transition. If the first thing your application is calling a format, then you should add the sentence "call cShowMain (0)" in the subroutine cetwin.b cetPreInit in the file.
The main window can also hide before showing a dialog. In that case, the dialog is visible and centered on the main window will remain invisible.
cShowStatusBar
This function can be used to display or hide the status bar. When the status bar is visible the window is automatically resized to fit the screen. The CALL statement syntax is:
cShowStatusBar call (status%)
Where:
state
is an integer that can be or 0 to hide the status bar to display or 1.
You should use this feature in all application programs especially if you have used the cStatusText. If this is the case, we recommend that you call this function from the cetwin.b file you are using. A case in which only needs to make a call when you want to hide is cShowStatusBar.

cShowToolBar

This function is used to display or hide the toolbar. When the toolbar is visible window is automatically sized to fit the screen. The CALL statement syntax is:
cShowToolBar call (status%)
Where:
state
is an integer, 0 to hide the toolbar or 1 to view.
Note that if you want to be visible on the toolbar when you start your program, it is recommended to call this function from the cetwin.b file you are using. If you created a new toolbar with a call to CToolBar, your toolbar will be displayed automatically. Otherwise, it will display the default toolbar.
cShowVersion
This routine can be used to display the version of the Application Development System, each component of the system running in a message box. The statement syntax is:
call **cShowVersion**
The version of each component execution system (cetcab.lib, ctb32.lib, cetcmpt.lib and brtm.exe) will be displayed in separate windows. The last window display can be hidden by application window since brtm is a separate task. If you can not see the messages, move or minimize the application window.

cSizeAppWindow

This function can be used to size the main application window to a certain size. The syntax of the call is:
cSizeAppWindow call (x%, y%)
Where:
x, y
is the new height and width in units of pixels.
The easiest way to determine the width and height of the application is to open a dialog box to the desired size and then look at its width and height properties. Remember that the maximum default size, character-based windows, is the size required to display 80 * 25 characters using the current font.

cStatusText

This function can be used to replace the default message of the status bar. The syntax is:
cStatusText call ($ msg)
Where:
msg
is the message string to display.
For example:
CALL cStatusText ("Enter the name of the Customer")

When the status bar is displayed for the first time, the default message is "Ready" (Ready / Standby). The messages for different menu items and buttons on the toolbar will also appear on the status bar. When the mouse cursor leaves the menu or toolbar, the default message will be displayed again.
At any time you can change the default message with a call to your program cStatusText to always display the appropriate message to the user.

cStringInputQ

This function can be used to enter a string in the input buffer. The result is exactly as if the characters were entered from the keyboard. The syntax of this statement is:
cStringInputQ call (string $)
Where:
chain
is the character string to be placed inside the keyboard input queue.
For example:
CALL cStringInputQ ("Hello World")
See cInputQ function for a similar function that lets you insert characters ESC within the input buffer.

CToolBar

The default toolbar tools w/32 CET Apl. Builder has two buttons as "shortcuts" to quickly access the elements of source selection and menu items Help-About. This toolbar is defined by the following line in the section cetuser.rc Bitmap file:
IDR_MAINFRAME BITMAP MOVEABLE PURE "CETTOOL.BMP"
The graphical buttons (iconografiados) are stored within the cettool.bmp file in the \ cetlib \ inc if you used the default path during installation. You can not change the default toolbar, but if you can add new toolbars and CToolBar call to replace the toolbar from your application.
To design your own toolbar, copy the default toolbar in your local directory with a new name. For example:
copy \ cetlib \ inc \ cettool.bmp \ tst \ mitool2.bmp
Now, use the program to modify the file Imagedit mitool2.bmp. A bitmap toolbar buttons can have up to twenty. Each button within the bitmap is 15 dots high and 16 dots wide, therefore the bitmap of a toolbar with two buttons 15 should be high by 32 dots wide. Three buttons would be 15x48 and so on.
When you need to modify the bitmap file, add a line within within cetuser.rc to use the new toolbar.
IDR_TOOLBAR2 BITMAP MOVEABLE PURE "MITOOL2.BMP"
You can have up to 50 toolbars using predefined identifiers ranging from IDR_TOOLBAR2 up ID_TOOLBAR50 adding them to the file. Rc. Note that the identifier IDR_TOOLBAR1 is reserved for the default toolbar called IDR_MAINFRAME.
After the new toolbar has been defined in the file cetuser.rc (and recompiled the file) and can make a call to CToolBar to load the toolbar and menu items associated with the buttons. (This can be done in the initialization routine of your BASIC program or routine within the framework cetPreInit Windows)
The CALL statement syntax is:
CToolBar call (bmid% Num% MATADDROF (btns%))
Where:
Bmid
is the ID number of the bitmap resource. The default toolbar as IDR_MAINFRAME has an ID number from 1 (bmid% = 1) and is associated with the file cettool.bmp. Use of bmid% = 2 to bmid% = 50 to reference defined toolbars from IDR_TOOLBAR2 up IDR_TOOLBAR50.

Num
is the number of buttons on the toolbar, including separators, this argument defines the number of array elements btns%.
btns
is an integer array of 50 elements representing menu ID values (from ID_WINSUB1 up ID_WINSUB250) to be associated with the buttons. The first element of the array is always 1. If used OPTION BASE 1 element 0 is ignored.
If you consider that each button is a quick way to specify a menu item, it is easy to understand that each button (within the matrix btns) has to be associated with an identifier, just as you MENUITEM serves to define the individual elements associated with a menu.
A menu specific ID is associated with a corresponding button in the integer array to the position of the button in the bitmap file. For example if your toolbar has three buttons, the dimension of the array will have a size of 3. The first button may be associated with the menu item ID_WINSUB12, the second and the third ID_WINSUB2 ID_WINSUB34 for assigning the following values:
btn% (1) = 12 \ btn% (2) = 2 \ bth% (3) = 34
There are six special values of ID you can use. These are:

0 is a value known as SEPARATOR used to add space between buttons.
500 is the ID associated with menu Source (ID_FONTS_SELECT)
501 ID associated with the menu Help-About (ID_APP_ABOUT)
502 ID associated with the Exit menu (ID_APP_EXIT)
503 ID associated with the Toolbar menu (ID_VIEW_TOOLBAR)
504 ID associated with the State Bar menu (ID_VIEW_STATUS_BAR)
The separator does not consume any buttons, but adds separating buttons so they can be grouped into roles. If using SEPARATOR, increase the number of buttons (Num%) with the number of these separators used as this argument defines the size of the matrix btn%.
The sample code shown below loads a new toolbar with three buttons and a separator between the second and third buttons:

```
REM Load a toolbar with 3 buttons and separator
REM between buttons 2 and 3. Bmid argument 2% is associated
REM with IDR_TOOLBAR2 which should be added to the file. Rc
REM and associate it with a file. Bmp represent its buttons.
DIM btn% (4)
btn% (1) = 12 \ btn% (2) = 2 \ btn% (3) = 0 \ btn% (4) = 34
REM Hides the tool bar before replacing.
CALL CToolBar (2.4, MATTADDROF (btn%))
```

Note: See chapter Framework Windows for information on how to make a toolbar appear every time you run your program.

cWaitCursor

This routine is used to display a cursor type `hourglass' (waiting). This normally is used to inform the user to wait while it is running another operation. The syntax is:

```
cWaitCursor call (expected%)
```

Where:
waiting
is an integer value, within which one shows an hourglass cursor and -1 hides it, any other value will be ignored or cause an error.

cWinExec

This routine can be used to run a Windows application. Note that this function transfers control to the specified program (as does the judgment CSI). As soon as the call is made cWinExec, will run the next program statement. See cCreateProcess and cProcessStat function to see how to start a process and verify its status to determine if another operation can be developed, depending on the state.
The CALL statement syntax is:

```
cWinExec call ($ path, showing%)
```

Where:
road
specifies the full path name of the executable file. This argument exists so you can replace the default search using the PATH. Note that this argument only affect the executable. Windows programs only look for a file in the current directory unless you specify a full path. The search sequence is:
 The directory where the calling program is executed.
 The current working directory
 The \ Windows or \ Windows \ system
 The directories set in the PATH
 directories within the variable set B_LKPATH
Note that this argument only affects the executable. Windows programs are only searched in the working directory, unless you specify the full path.
show
specified will be displayed as the application.
For example:

```
CALL cWinExec ("notepad", 1)
```

The show argument determines how it will be displayed the application. Valid values are:
1 - in a normal window.
2 - in a window (icon) minimized.
3 - in a maximized window (full screen).
Once the application is finished, control returns immediately to the calling program. See function in programs W32DEMOS examples to see how it works, and could also use the cProcessStat.
Since CSH / CSI are designed to run MS-DOS commands, you can run Windows applications using CSI. (Currently it can be done under Windows NT, but not under Windows).

Also note that you can use to launch executables cWinExec DOS under Windows NT, but the argument show work differently. If a file is created. Pif with the property "windowed" then the values will be:
1 - As a normal window
2 - As a window (icon) minimized.
3 - As a normal window
The argument showing no effect on a DOS program that runs from a file. Bat. Refer to implementation notes in the appendix for more information on using the CSI sentence.
See Annex implementation notes, statements about the use of CSI.

CWallPaper

This routine can be used to display an image as wallpaper in the main application window. The image must be available in a format Windows bitmap (ie, BMP or DIB). The syntax is:
Call cWallPaper (path $, x%, y%, w%, h%)
Where:
$ Path
Is the full path name of the file. Dib or. Bmp to be displayed.
X%, Y%
Are the column and row of the upper left corner where the image should be displayed.
W% H%
Are the width and height of the image area to show.

8.2.1 Functions list

The following chart lists the functions w/32, with a brief description of its use.

Función	Operación
cAppTitle	Agrega el título a mostrar de la ventana de aplicación
cAppWindowInfo	Obtiene la localización y tamaño de la ventana de aplicación
cAppWindowTop	Mueve la ventana de aplicación a la parte de arriba
cAppWindowMax	Maximiza la ventana principal de la aplicación.
cAppWindowMin	Maximiza la ventana principal de la aplicación.
cAppWindowNormal	Restaura el tamaño de la ventana principal de la aplicación.
cCheckItem	Introduce o borra un botón de opción para un elemento de menú
cChoiceList	Visualiza una lista de opciones con elementos clasificados
cChoiceListNS	Visualiza una lista de opciones con elementos no clasificados
cCloseDIBWindow	Cierra una ventana DIP
cCreateProcess	Crea un proceso específico
cDIBInfo	Obtiene el alto y ancho de un gráfico DIB
cDIBWindowInfo	Obtiene la localización y tamaño de una ventana DIB
cDIBWindowTop	Desplaza una ventana DIB al principio del stack (pila) de ventanas
cDisplayInfo	Obtiene la resolución de pantalla y número de colores
cEnableDocking	Habilita una barra de herramientas `movible`
cEnableItem	Habilita o deshabilita un elemento de menú.
cEnableMainWin	Controla el estado de activación de la ventana de la aplicación.
cEnumPrinters	Obtiene información de las impresoras disponibles
cFontValues	Devuelve los valores de un fuente (font) seleccionado.
cForceExit	Sale del programa sin un cuadro de mensaje tipo Aceptar
cGetCursor	Obtiene las posiciones actuales de columna y fila del cursor.
cGetMouse	Obtiene la posición actual del ratón.
cInputQ	Introduce un código de tecla dentro la cola de entrada.
cLogin	Muestra un cuadro de diálogo login.
cMaxSize	Redimensiona la ventana a su tamaño máximo

cMessageBoxOkC	Visualiza un cuadro de mensaje con Aceptar/Cancelar
cMoveAppWindow	Desplaza la ventana de aplicación a una nueva localización.
cMoveDIBWindow	Mueve una ventana DIB a una nueva localización.
cNewMenu	Muestra un menú nuevo.
cOnWin32s	Chequea sobre qué tipo de Windows está rodando
cOpenBrowser	Muestra una ventana de selección de Abrir Archivo.
COpenDIBWindow	Abre una nueva ventana para mostrar un fichero de mapa de bits.
cProcessStat	Chequea el status de un proceso arrancado
cSaveBrowser	Muestra una Ventana tipo Salvar como.
cSelectFont	Cambia al fuente especificado.
cSetCursor	Fija el cursor para la próxima operación de entrada.
cShowMainWin	Hace visible o invisible la ventana principal de la aplicación.
cShowStatusBar	Esconde o muestra la barra de estado.
cShowToolBar	Esconde o muestra la barra de herramientas.
cShowVersion	Visualiza la versión de todos los componentes del sistema de ejecución.
cSizeAppWindow	Establece un tamaño determinado en la ventana principal de la aplicación.
cStatusText	Muestra texto sobre la barra de estado.
cStringInputQ	Introduce un cadena dentro del buffer de entrada.
cToolBar	Carga una nueva barra de herramienta de mapa de bits.
cWaitCursor	Visualiza o quita un cursor tipo reloj de arena.
cWinExec	Ejecuta una Aplicación Windows.

8.3 W/32 functions list

All built in C language functions including WINDOW CET for use in your applications listed in this apartado.Nótese that the names of the functions that are available on all products CET begin with the prefix 'B', while the functions solamante that are available on this product starting with 'c'.
In the table:
Gen. are generally useful functions.
Dlg. The functions to be used with dialogues.
Mez. The functions of several.

Función	Operación	Gen.	Dlg	Me
Bchdir	Cambia el directorio de trabajo actual.			♦
Bcopy	Copia el contenido de un fichero a otro.			♦
Bcreate	Crea (o limpia) un fichero específico de datos.			♦
Becho	Muestra los caracteres introducidos mediante funciones LINPUT e INPUT.			♦
Berase	Elimina un fichero determinado.			♦
Bfilestat	Determina si existe un fichero o directorio.			♦
Bgetcwd	Lee el directorio actual.			♦
Bgetenv	Lee el contenido de una variable de entorno.			♦
Bindinfo	información sobre un fichero indexado			♦
Bkbshift	Consigue el estado de letras específicas.			♦
Bnoecho	Desactiva el eco de caracteres en funciones LINPUT			♦

o INPUT.			
Bputenv Pone un valor en una variable de entorno.			♦
Brename Renombra el fichero especificado.			♦
Bsernum Lee el número de serie de la llave.			♦
cAddBoxContents Establece el contenido de una caja de Listado.	♦		
cAddComboContents Establece el valor actual de una caja Combo.	♦		
cAppTitle Agrega el título a mostrar de la ventana de aplicación			
cAppWindowInfo Obtiene la localización y tamaño de la ventana de aplicación			
cAppWindowTop Mueve la ventana de aplicación a la parte de arriba	♦		
cAppWindowMax Maximiza la ventana principal de la aplicación.	♦		
cAppWindowMin Maximiza la ventana principal de la aplicación.	♦		
cAppWindowNormal Restaura el tamaño de la ventana principal de la aplicación.	♦		
cCbCopy Copia una cadena de caracteres al Clipboard	♦		
cCbCanPaste Comprueba el contenido del Clipboard	♦		
cCbEmpty Limpia el contenido del Clipboard	♦✔		
cCbPaste Pega una cadena de caracteres desde el Clipboard	♦		
cCheckItem Introduce o borra un botón de opción para un elemento de menú	♦		
cChoiceList Visualiza una lista de opciones con elementos clasificados	♦		
cChoiceListNS Visualiza una lista de opciones con elementos no clasificados	♦		
cCloseDIBWindow Cierra una ventana DIP	♦		
cComboAddString Añade un alfanumérico en una caja Combo.		♦	
cComboDelString Elimina un alfanumérico de una caja Combo.		♦	
cComboInsString Inserta un alfanumérico en una caja Combo.		♦	
cComboSetChoice Establece la selección por defecto de una caja Combo.		♦	
cCreateProcess Crea un proceso específico.	♦		
cDefaultBtn Cambia el botón de comando por defecto.		♦	
cDIBInfo Obtiene el alto y ancho de un gráfico DIB.	♦		
cDIBWindowInfo Obtiene la localización y tamaño de una ventana DIB.	♦		
cDIBWindowTop Desplaza una ventana DIB al principio del stack (pila) de ventanas .	♦		
cDisplayInfo Obtiene la resolución de pantalla y número de colores	♦		
cDlgTitle Establece un título para el diálogo.		♦	
cEnableCtrl Habilita o deshabilita un control.		♦	
cEnableDocking Habilita una barra de herramientas `movible`	♦		
cEnableItem Habilita o deshabilita un elemento de menú.	♦		
cEnableMainWin Controla el estado de activación de la ventana de la aplicación.	♦		
cEndDlg Establece el CMDID% para finalizar el diálogo.		♦	
cEnumPrinters Obtiene información de las impresoras disponibles	♦		
cFontValues Devuelve los valores de un fuente (font)	♦		

	seleccionado.			
cForceExit	Sale del programa sin un cuadro de mensaje tipo Aceptar	♦		
cFormBtnClick	Envía un evento a una forma activa.		♦	
cGetCheck	Obtiene el estado de una caja de comprobación o de un radio botón.		♦	
cGetCombo	Obtiene la selección de una caja Combo.		♦	
cGetCtrlText	Obtiene el contenido de un control estático (una etiqueta)		♦	
cGetCtrlValue	Lee el contenido de un control interactivo.		♦	
cGetCursor	Obtiene las posiciones actuales de columna y fila del cursor.	♦		
cGetEditText	Obtiene el valor de un control de edición.		♦	
cGetBoxSelection	Obtiene el valor del ID seleccionado de una caja de listado.		♦	
cGetBoxString	Obtiene el string seleccionado de una caja de listado.		♦	
cGetLimitEditText	Consigue el valor máximo de caracteres a introducir en un editor de texto.		♦	
cGetMouse	Obtiene la posición actual del ratón.	♦		
cGetRadioGroup	Obtiene el radio botón seleccionado de un grupo de radiobotones.		♦	
cGetScrollPos	Obtiene la posición actual de una barra de Scroll		♦	
cInputQ	Introduce un código de tecla dentro la cola de entrada.	♦		
cLBoxAddString	Añade un string al final de una caja de listado.		♦	
cLBoxDelString	Borra un string de caja de listado.		♦	
cLBoxInsString	Inserta un string en una caja de listado.		♦	
cLBoxSetSel	Determina el string seleccionado en Limita los caracteres de una caja de listado.		♦	
cLBoxTabStops	Establece el número de paradas en el movimiento del tabulador.		♦	
cLBoxBoxTopIndex	Determina el primer elemento que debe aparecer en la lista.		♦	
cLimitComboText	Limita los caracteres de una caja Combo.		♦	
cLimitEditText	Limita los caracteres de un control de edición.		♦	
cLinputOpts	Establece las opciones del -Zi	♦		
cLogin	Muestra un cuadro de diálogo login.	♦		
cMaxSize	Redimensiona la ventana a su tamaño máximo	♦		
cMessageBoxOkC	Visualiza un cuadro de mensaje con Aceptar/Cancelar	♦		
cMoveAppWindow	Desplaza la ventana de aplicación a una nueva localización.	♦		
CMoveDIBWindow	Mueve una ventana DIB a una nueva localización.	♦		
cNewMenu	Muestra un menú nuevo.	♦		
cOnWin32s	Chequea sobre qué tipo de Windows está rodando	♦		
cOpenBrowser	Muestra una ventana de selección de Abrir Archivo.	♦		
COpenDIBWindow	Abre una nueva ventana para mostrar un fichero de mapa de bits.	♦		
cPassword	Pide clave de acceso en cualquier parte del programa	♦		

cProcessStat	Chequea el status de un proceso arrancado	♦		
cSaveBrowser	Muestra una ventana tipo Salvar como.	♦		
cSelectFont	Cambia al fuente especificado.	♦		
cSendBtnClick	Envía un evento de pulsación de botón.		♦	
cSetCaret	Establece la posición del cursor en un control de edición o en una caja combo.		♦	
cSetCheck	Establece el estado de una caja de comprobación o de un radio botón.		♦	
cSetCtrlText	Establece el contenido de un control estático (etiquetas).		♦	
cSetCtrlValue	Establece el contenido de un control interactivo.		♦	
cSetCursor	Fija el cursor para la próxima operación de entrada.	♦		
cSetEditText	Establece el contenido de un control de edición.	♦		
cSetFocus	Pone el focus en un control determinado.	♦		
cSetLimitEditText	Establece un valor máximo de caracteres a introducir en un editor de texto.		♦	
cSetScrollPos	Establece la posición de la barra de Scroll.	♦		
cSetScrollPage	Establece la 'pagina' de valores de una barra de scroll.	♦		
cSetShowRange	Establece los valores mínimo y máximo de la barra.	♦		
cSetShowCtrl	Esconde o muestra un control.	♦		
cSetShowEditWrap	Activa o desactiva la inclusión de caracteres de saltos-de-línea en un control de edición multilínea.	✔		
cShowMainWin	Hace visible o invisible la ventana principal de la aplicación.		✔	
cShowStatusBar	Esconde o muestra la barra de estado.		✔	
cShowToolBar	Esconde o muestra la barra de herramientas.		✔	
cShowVersion	Visualiza la versión de todos los componentes del sistema de ejecución.		✔	
cSizeAppWindow	Establece un tamaño determinado en la ventana principal de la aplicación.		✔	
cStatusText	Muestra texto sobre la barra de estado.		✔	
cStringInputQ	Introduce un cadena dentro del buffer de entrada.		✔	
cToolBar	Carga una nueva barra de herramienta de mapa de bits.		✔	
cWaitCursor	Visualiza o quita un cursor tipo reloj de arena.		✔	
cWinExec	Ejecuta una Aplicación Windows.		✔	

Chapter 9 BASIC subroutines

9.1 Introduction

Development System Win/32 CET application lets you compile a BASIC program that calls external BASIC subroutines. This feature allows you to integrate a set of BASIC programs into a single executable. The advantages of this system are:

- The execution is much faster, so all operations are eliminated and LINK CHAIN.
- The size of the resulting executable, although large, is much smaller than that of the individual BASIC programs it replaces.
- Some developers have found that calls to external subroutines compiled is the most natural way to program. Using routines (instead of chains and links) is similar to the method that follows C, Pascal, COBOL and other languages to write their applications.
- It is much easier to encode structured programs
- Normally used routines can be stored in external files, then called as needed by typing the name of the subroutine in the main program. This feature saves considerable time in development time since the code does not have to be constantly rewritten.
- It is much easier to maintain. A BASIC subroutine can be modified to accommodate some new requirements. Simply recompile programs using the subroutine.

BASIC subroutines use a series of new sentences. This chapter will try to document these statements and note how they affect the operation of a few sentences CET BASIC conventional.

9.2 Compiling programs that call BASIC Subroutines

Programs that call subroutines are compiled BASIC OBW32 options and syntax using 'normal'. The only difference is that each module can compile simultaneously with a single command. For example:
OBW32-or main principal.b sub1.b sub2.b
It is also possible to compile each module separately and link them together as:
OBW32-c sub1.b
OBW32-c sub2.b
OBW32-or main principal.b sub1.obj sub2.obj
When working with the integrated development environment to develop your application, automatically compile W32APBLD being unnecessary to recall the grammar of this command. By default, always use the-c flag in all BASIC source programs.

9.3 BASIC subroutines

BASIC subroutine statements differ from conventional CET BASIC statements on two things:

• Each statement begins with a dollar sign ($).

• With the exception of $ $ EXIT and CLEAR, judgments are not enforceable. Instead, report the compiler to treat the module and the sentences that constitute a particular way.

The sentences are:

$ MAIN Indicates this module is the main routine

$ SUB Indicates this module is a subroutine

$ GLOBAL Provides a list of variables that are global to all subroutines.

$ Clear clears all variables and dimensioned tables inside the subroutine, except those defined with $ GLOBAL.

$ EXIT Exits a BASIC subroutine.

9.4 $MAIN sentence

The $ MAIN must be the first statement in the main module. The syntax is:
$ MAIN
Since only one module (the main module) that use the $ MAIN judgment of all modules that form the application. The main module is the segment of the program under which the program starts.

9.5 $SUB sentence

BASIC subroutine begins with a statement defining $ SUB mode subroutine that can be called from another BASIC subroutine or program. The syntax is:
$ SUB-name module
Module-name is case sensitive and sensitive, so TestSub, TestSub and TestSub are all different. For example:
$ SUB mysub
$ SUB WriteRpt
BASIC Subroutines are invoked with the BASIC CALL statement.

9.6 $GLOBAL sentence

Data from BASIC subroutines can be shared through one of the following methods:

1. Any variables referenced within the same program or file. B refers to the same variable. In other words, all variable names are global to BASIC subroutines that reside within the same program.
2. Any variable that is being used in different modules are separate sources unless they are defined by $ GLOBAL statement. This is true even with variable of the same name, or you can define local variables with the same name in separate source modules.

This feature of the manager `names' lets you write and compile separate BASIC subroutines that bind within an executable, without worrying about possible conflicts between variable names.
When using $ GLOBAL statement, it must be preceded by $ MAIN in the main module or $ SUB statement in subroutines, otherwise the compiler would give an error of external inconsistencies. The statement syntax is:
$ GLOBAL variables-list
The variable-list is a list of BASIC variable names that refer to the same variable in all modules. The list of variables is composed of a simple scalar variable name and / or names of matrices (tables) preceded by the MAT keyword.
If you are going to define several variables to be shared, they can be specified in different lines $ GLOBAL or all in one line with commas separating variable names. For example, a $ GLOBAL typical sentence would be:
$ GLOBAL GVAR, MAT MYARRAY, ANYSTG $, I%, MAT XSTG $
Note that a table should be sized by either DIM or COMMON statements before moving to a BASIC subroutine.
For example, consider the main program and the subroutine following three variables in this shared ($ GLOBAL). Values and Table CNT% ARRAY% are allocated in the main routine and are passed to the subroutine SumArray, which will add the number of elements CNT% (10 elements in this case) and returns the result in AnSWR%. These variables are exactly the same in both programs and occupy the same addresses.

```
REM
REM MAIN.B filed under Main Program
REM
$ MAIN
$ GLOBAL CNT% ARRAY% MAT, AnSWR%
DIM ARRAY% (10)
FOR I% = 1 TO 10
ARRAY% (i%) = I%
NEXT
CNT = 10%
CALL SumArray
PRINT "The answer is:"; AnSWR%
END
REM
REM Subroutine SUM.B file with the file
REM
$ SUB SumArray
$ GLOBAL CNT% ARRAY% MAT, AnSWR%
AnSWR% = 0
FOR I% = 1 TO CNT%
AnSWR AnSWR% +% = ARRAY% (I%)
NEXT
$ EXIT
```

To summarize, the most important thing to remember when using variables within programs that call BASIC subroutines are:

- Each variable is within a $ GLOBAL list, is the same variable in each module that uses it.
- If a variable is not defined within a list of variables $ GLOBAL (or common) must be initialized each time you enter or exit a subroutine using a sentence $ CLEAR.
- Each variable $ GLOBAL used in a BASIC subroutine must also be declared in the main program.
- The $ GLOBAL must immediately follow the rulings and $ $ MAIN SUB.

9.7 $CLEAR sentence

Normally, variables are initialized when in a subroutine. $ CLEAR can be used when needed and initialize all variables dimensioned arrays that are not defined as GLOBAL or $ COMMON. The statement syntax is:
$ CLEAR
The reason to initialize all variables from the fact that this is what normally happens when a CET BASIC program runs for the first time. Besides variables in separate programs are different (unless they are in COMMON), even though they have the same name. Programs must take this into account. $ CLEAR program provides for a `new 'BASIC subroutines used to combine a number of separate programs before using statements CHAIN, LINK or RUN, operate the same way.

Unless a variable is in a list $ GLOBAL (or common), $ CLEAR the following effect when entering a subroutine:
• Scalar variables (no matrix) integer and real are zeroed.
• Scalar variables are loaded with alphanumeric null string ("").
• Arrays are cleaned, that is, each array is initialized to a one-dimensional table with 10 elements, each of which is zero or nil, depending on the data type.
Use of this judgment is critical, because if we do not use local variables within a subroutine, to which no value is assigned (by an assignment statement) its contents can be unpredictable, since the system does not guarantee initializing them upon entering the subroutine, unless you use the $ statement CLEAR.

9.8 $EXIT sentence

$ EXIT is the only one of the new sentences that is executable. When is $ EXIT during program execution, control returns to the module that called it, ie the statement following the CALL. Internally this works as a RETURN statement. A $ EXIT default be implemented at the end of a module $ SUB in the case does not exist. The statement syntax is:
$ EXIT
The statements END, QUIT and STOP can not be used instead of $ EXIT. If such a statement is encountered during the execution of a program, within any module, the entire program terminates immediately.

9.9 Converting existing applications in BASIC CET

BASIC subroutines are often used to convert applications are usually composed of a number of separate programs. The steps to do this include:

1. Determine the 'tree' implementation that should monitor their implementation. For example, suppose you have CHAINs main program menu or links to other programs.
2. Add $ MAIN to the first line of your main program (menu) which use CALL statements to call the next level (submenu) program (BASIC subroutines now), which depending on conditions resulting implementation of each of these in submodules (eg. answers user, a fixed sequence or some other method of selection).
3. Programs that previously shared COMMON variables should be sited in a list $ GLOBAL. Remember to use the keyword MAT prefix in front of the arrays (tables) to include in $ GLOBAL. (Also make sure that the $ GLOBAL variables are dimensioned.
4. Replace all statements and LINK CHAIN by the CALL statement. The precise way in which to do this and the location of the CALL statement depends on the structure of their "execution tree".
5. $ Use CLEAR to clear all variables and arrays with undefined $ COMMON GLOBAL or before entering (or exiting) the subroutine and not keep loaded during a CALL previous values.
6. Modify programs to return to the calling program automatically with a $ EXIT statement (in our example would change $ EXIT by Chaim ("menu"). The $ EXIT ensure that the control is automatically returned to the program that called it using a CALL statement.
7. Remember all sentences end with a RETURN GOSUB before it is executed a $ EXIT, but the program passes control to the last statement that is executed GOSUB.

8. You may continue using CHAIN, LINK or RUN between programs / main modules. In these cases, you can still use the COMMON statement to switch between programs, variables and arrays. If you use a common statement that would be used only in the $ MAIN module. To share these variables (COMMON or not) with a BASIC subroutine should use the $ GLOBAL statement.

9.10 A conversion example of an application BASIC CET

The following example illustrates what needs to be done to make the main program with LINK C_HAIN and other programs submenu. Consider the following programs CET BASIC:

BASIC Program

```
COMMON CONAME$
PRINT "Seleccione Programa"
PRINT "1. Contabilidades Acreedoras"
PRINT "2. Contabilidades Deudoras"
PRINT "3. Libro Mayor"
PRINT "4. EXIT"
INPUT ANS%
SELECT ANS%
    CASE 1
     CHAIN "ar"
    CASE 2
     CHAIN "ap"
    CASE 3
     CHAIN "gl"
    CASE 4
     QUIT
CEND
```

Developed Program menu.b menu.b

```
$MAIN
$GLOBAL CONAME$
 WHILE 1
        PRINT "Seleccione Programa"
        PRINT "1. Contabilidades Acreedoras"
        PRINT "2. Contabilidades Deudoras"
PRINT "3. Libro Mayor"
 PRINT "4. EXIT"
 INPUT ANS%
 SELECT ANS%
 CASE 1
        CALL ar
 CASE 2
                CALL ap
        CASE 3
                CALL gl
        CASE 4
QUIT
 CEND
 WEND
```

Programa BASIC ar.b

```
COMMON CONAME$
<Línea Principal de ar>
CHAIN "menu"
```

Programa Convertido ar.b

```
$SUB   ar
$GLOBAL CONAME$  \  $CLEAR
 <Línea Principal de ar>
 $EXIT
```

Programa BASIC ap.b

```
COMMON CONAME$
<Línea principal de ap>
CHAIN "menu"
```

Programa Convertido ap.b

```
$SUB    ap
$GLOBAL CONAME$ \ $CLEAR
 <Línea Principal de ap>
 $EXIT
```

Programa BASIC gl.b

```
COMMON CONAME$
<Línea Principal de gl>
CHAIN "menu"
```

Programa Convertido gl.b

```
$SUB    gl
$GLOBAL CONAME$ \ $CLEAR
 <Línea Principal de gl>
 $EXIT
```

In these examples, very few changes have been made to convert the conventional application CET BASIC. None of the codes in the <line principal> programs (which may include hundreds of lines of BASIC) had to be modified. The $ EXIT is used to replace each of the explicit statements CHAIN `s back to the main menu and the COMMON statement is replaced with a $ GLOBAL. Being added a $ CLEAR to clear all other variables and arrays in the subroutine.

9.11 Differences cet basic conventional programs

BASIC subroutines are designed to allow the compilation of the modules separately and be linked with an executable while requiring only small changes to the existing application of CET BASIC.

During execution, there are some statements CET BASIC acting differently when they were compiled into the program of separate modules. The following sections deal with these differences:

9.11.1 Error handling

Importantly, each BASIC subroutine performs its own error handling. Each subroutine where an error could occur a sentence begins with "ON ERROR GOTO". This also applies to the interrupt processing and the key locking system.

9.11.2 RESTORE y READ with the DATA sentences

Normally the READ statement, when not accompanied by a file number, read through a list of data specified by the DATA statements. Each READ hereinafter "use" the next item in this list consists of data each program DATA statements.
The pointer (pointer) of the next data item can be changed using RESTORE. This statement can "turn back" to "pointer" to the first DATA statement or another DATA statement as specified line number.
When using the BASIC subroutines, observe the following behavior.

- A READ statement reads only through the data in the module in which it is located.
- A RESTORE statement can only refer to the DATA statement in the same module.

9.11.3 COMMON sentences

The COMMON statement is a powerful executable statement, which adds a set of variables, to the existing list of common variables. During a CHAIN or LINK, this list of common variables will be "matched" with the variable list (s) statement (s) of the program called COMMON.
When using the BASIC subroutines, it is advisable to use the COMMON statement except in the main module ($ MAIN). If you want to share variables with other modules use the SQL $ GLOBAL.

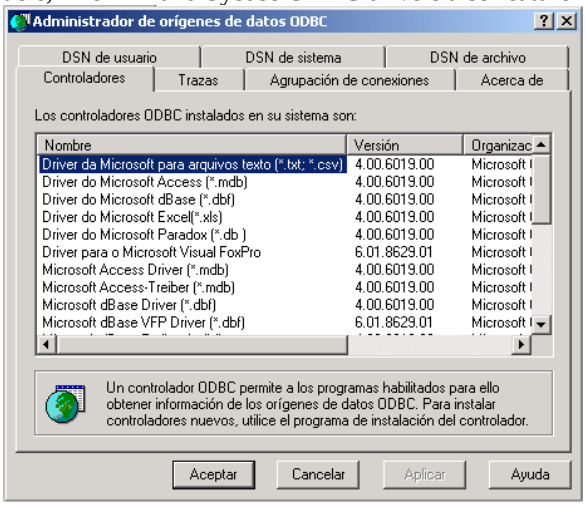
9.11.4 User defined functions

A user-defined function, specified by the DEF statement is recognized only by the module in which it is defined.

Chapter 10 ODBC support

10.1 Introduction

The connectivity of open database (ODBC) is a standard created by Microsoft to allow transparent access from applications to a wide range of databases and file management systems.
There ODBC drivers in Microsoft Windows within the file handle (eg dBase, Btrieve), database managers (eg Fox Pro, Paradox and MS Access) and SQL servers as Watcom SQL, and SQL Gupta. Relational databases such as MS SQL Server, Oracle, Informix and Sybase ODBC drivers also feature for desktop connectivity


Administrador de orígenes de datos ODBC
DSN de usuario
DSN de sistema
DSN de archivo
Controladores
Trazas
Agrupación de conexiones
Acerca de
Los controladores ODBC instalados en su sistema son:
Nombre
Versión
Organizac
Driver da Microsoft para arquivos texto (*.txt; *.csv) 4.00.6019.00
Driver do Microsoft Access (*.mdb) 4.00.6019.00
Driver do Microsoft dBase (*.dbf) 4.00.6019.00
Driver do Microsoft Excel(*.xls) 4.00.6019.00
Driver do Microsoft Paradox (*.db) 4.00.6019.00
Driver para o Microsoft Visual FoxPro 6.01.8629.01
Microsoft Access Driver (*.mdb) 4.00.6019.00
Microsoft Access-Treiber (*.mdb) 4.00.6019.00
Microsoft dBase Driver (*.dbf) 4.00.6019.00
Microsoft dBase VFP Driver (*.dbf) 6.01.8629.01
Un controlador ODBC permite a los programas habilitados para ello obtener información de los orígenes de datos ODBC. Para instalar controladores nuevos, utilice el programa de instalación del controlador.
Aceptar
Cancelar
Aplicar
Ayuda

To use ODBC w/32 within a program must have correct ODBC drivers installed on your system. You can get these ODBC drivers in various ways, from different sources. Often included with the product (for example, the ODBC driver for Microsoft Access is within the Access albums).
Once you have the correct ODBC drivers installed and the type of source files configured according to the documentation for the driver, you can access these files from within the program source w/32.
Standard statements such as OPEN, CLOSE, READ, MAT READ, WRITE, WRITE MAT, READNEXT, READNEXT MAT, MAT READPREV READPREV and may be used to access files through ODBC source.
All the values are written / read from the source files are alphanumeric (w/32 convert all alphanumeric values from / to native types of source files automatically).
Note that B_ODBCERR environment variable must be updated to 1 in W32APBLD.w32 file to display messages from any ODBC error which may occur in implementation.

10.2 Opening a file

To open a file in ODBC source, use the OPEN statement with the INDEXED option, either in form or in the form INPUT UPDATE. The filename argument determines which type of ODBC source file will open and can be specified in one of two formats:
ODBCxx;
The first format brings up a dialog box so that the user can select from any of the source files that are currently configured. The xx within ODBCxx is either a SS for a "snapshot" or DS for a "dynaset". A "snapshot" contains records that are valid when selecting the first time and then not kept current, that is, functions as if labor with a copy. Mode "dynaset" maintains consistency with existing source files by checking if the record is still present and taking a new record otherwise. A "snapshot" is faster to use and does not need to be continually accessing the source files.

```
ODBCxx; DSN = "<Name Source> files;"
```

The second format opens the name of a specific source file that was created when you configured the source file through ODBC Administrator. For example, the following statement must open a source file called "Test Data" to read and write on Channel 5 in snapshot mode:

```
OPEN # 5: "ODBCSS; DSN = Test Data" UPDATE INDEXED
```

When you open a file using the explicit name, the name must end with a semicolon sign (;). Otherwise you will see a runtime error "File not Found" (file not found).

10.3 Reading a file

Once the OPEN statement connects to the source file, the key field of the READ statement is used to obtain information specific to the source file and access records. Among the key may use any of the following reserved words or may be an SQL SELECT statement.

TABLES Returns all table names of type "table" that are found within the source file. Two values are returned with each table name and type.
ALLTABLES Returns all the names of the tables, including system tables. Two values are returned for each table the name and type.
TYPES return information about supported data types. The five types of return values are:
1. Native type data base.
2. The type associated SQL.
3. The precision of the column.
4. If the column is case sensitive or not.
5. If the column is searchable or not.
6. If the column is not a column or counter (auto-increment).
7. The column prefix.
8. The column suffix.
9. The creation parameters (CREATE PARMS) column
The information returned by TYPES is only required for advanced operations on the source files, as add new tables or columns.
The following example shows how to use the keyword TYPES. (It is assumed that the source file supports up to 30 types).

```
dim type $ (30.9)
mat read # 1, "types": type $
print
print "Data Types"
print
For i% = 1 to 30
if type $ (i%, 1) = "" then goto endtype
print type $ (i%, 1) + "(" + type $ (i%, 2) + ")" + "(" + type $ (i%, 3) + ")" + "(" + type $ (i%, 4) + ")" + "(" +
type $ (i%, 5) + ")"
print "(" + type $ (i%, 6) + ")" + "(" + type $ (i%, 7) + ")" + "(" +
type $ (i%, 8) + ")" + "(" + type $ (i%, 9) + ")"
wait
next i%
endtype:
```

COLUMNS: Returns all table names of the columns in the specified table. Three values are returned for each column: the name, the precision of the field, the field type.
Requires REQUERY source files to ensure that all records are updated. No information is returned to the program.
A SQL SELECT statement can also be used to select a number of columns in a specific table. The SELECT statement can contain many options (Maximum length of 2048 characters). The simplest form is:
"SELECT FROM <table> columns> name>": <numcol%>
where
Column name
is either a list of column names separated by the comma (,) or an asterisk "*" to select all columns.
table
is the name of the table to which you are accessing.
Numcol
is the number of columns that was selected.
MAT READ to not use a SELECT statement, the correct usage is:
READ # 5, "□ SELECT FROM Authors": NumCols%
Added a new keyword, "RESULTCOLUMS" for READ statements. After being successfully executed a SELECT statement, you can use this keyword to get a description of colomnas have been selected. Column returns three values: the name, the precision t the type of the column. For example:

```
READ # 1, "SELECT * FROM SECTION": NumCols%
DIM $ col (NumCols%, 3)
```

MAT READ # 1, "RESULTCOLUMNS": col $
After any SELECT statement, will be positioned at the beginning of the recordset. We have added four new keywords to the READ statement to improve mobility through the records:
MoveFirst is positioned before the first record (the first record becomes READNEXT)
MoveLast is positioned behind the last record. (READPREV returns to the last record)
MOVE: num Moves to a position relative to the current position. Negative values mean travel backward. A zero value causes no change. EOF is returned if moving backward principle is reached or if moving forward end is reached.
Starus Returns an index of the current record. The first record is record 0. If EOF will return the value -1. If you could not determine the index return -2.
Note that if you make a MoveLast and then does MovePrev or MOVE: -1, the current record is the last record, but because ODBC STATUS return -2 not yet know the index of the last record.
The total number of records can not be determined by more than one pass (counter) through all of them, or moving to the last record and using STATUS. If you have developed READNEXT operations from the beginning, then STATUS can determine the number of the current record. For example:

```
READ # 1, "any SELECT statement": $ NumCols
IF EOF (1) THEN ..... \ REM Always check for EOF on a SELECT
WHILE NOT EOF (1)
READ # 1, "MOVE: 1": $ dummy
WEND
READ # 1, "STATUS": LASTREC%
```

For example, the SELECT statement that follows, as a key field for READ, can be used to find the two columns "Name and Comments" in a table called "Authors".

10.4 Updating a file

The WRITE statement is used to add, update or delete records. If the WRITE operation fails, then an EOF condition.
To delete or update a record, first used READNEXT / READPREV to position the pointer on the appropriate record. The value of the key field for the WRITE statement can have three values: "UPDATE", "DELETE" or "ADDNEW".

10.4.1 Updating a column

To update the Comment column of the Author "Joe Smith" with "new comment" run what comes next:

```
dim result$ (2)
while 1
    mat readnext #5,"": result$
    if eof (5) then goto notfound
    if result$ (1) = "Joe Smith"
            result$ (2) = "a new comment"
            mat write #5,"update": result$
            goto done
            ifend
    wend
notfound:
    print "Couldn't find Joe Smith"
done:
```

10.4.2 Deleting a record

To delete the whole record "Joe Smith":

```
dim result$ (2)
while 1
    mat readnext #5,"": result$
    if eof (5) then goto notfound
    if result$ (1) = "Joe Smith"
            write #5,"delete": “
            goto done
            ifend
```

```
    wend
notfound:
    print "Couldn't find Joe Smith"
done:
```

Note that writing an alphanumeric vacuum. Although not used, but the compiler will generate an error if the WRITE statement is worthless to write.

10.5 An example program

The following program shows how you can choose by Browse, source file, select the table in columns with which they will work, and take the data selected for display.
Note that B_ODBCERR environment variable can be set to one in W32APBLD.w32 file to remove the ODBC error messages that may occur.

```
REM Permite hasta 20 nombre de tablas y columnas.
dim tables$ (20,2), col$ (20,3), rslt$ (20)
REM Abre un snapshot en forma de entrada. Escoge los ficheros
REM fuente en el momento de ejecución.
open #1: "ODBCSS;", input indexed
tables:
    REM Obtiene las tablas definidas para este tipo de datos.
    mat read #1, "TABLES": tables$
    REM Chequea el  EOF.   (EOF en las Tablas es un error.)
    if eof (1)
            msgbox "No tables defined" \ quit
            ifend
    REM Saca por pantalla las tablas en estos ficheros fuente.
    print \ print "Available Tables:" \ print
    for I% = 1 to 20
            if tables$ (i%,1) = "" then BREAK
            print TAB (8);tables$ (i%,1) + "  (" + tables$ (i%,2) + ")"
            next i%
    REM Deja que el usuario escoja que tabla desea trabajar.
    print \ print \ print "Select a Table name  (or Q to quit):";
    linput tname$
    if tname$ = "Q" then quit
    mat read #1, "COLUMNS:"+tname$: col$
    REM Chequea si ha llegado al final del fichero EOF.   (EOF en las
    REM columnas es un error).
    if eof (1)
            msgbox "No columns in the selected table"
            goto tables
    ifend
    REM Saca por pantalla las columnas de esta tabla.
    print \ print \ print "Available Column names:" \ print
    for I% = 1 to 20
            if col$ (i%,1) = "" then BREAK
            print TAB (8);col$ (i%,1) + "  (" + col$ (i%,2) + ")" + "  ("
+ col$ (i%,3) + ")"
    next i%
    REM Deja al usuario escoger con que columnas trabajar.
    print \ print \ print "Select Column names  (comma                    separated)        for
retrieval  (or * for all):";
    linput cname$
    mat read #1, "SELECT "+cname$+" FROM "+tname$: a$
    REM Comprueba si ha llegado al final del fichero  EOF.   (EOF
    REM sobre SELECT significa que no se eligió ningún registro.)
    if eof (1)
            msgbox "EOF on SELECT" \ goto tables
```

```
ifend
REM Saca por pantalla las columnas seleccionadas del registro
REM elegido.
while 1
        print \ print
        mat readnext #1, key$: rslt$
        if eof (1) then BREAK
                for I% = 1 to 10
                        if rslt$ (i%) = "" then BREAK
                        print TAB (8); \ print rslt$ (i%)
                next i%
                wait
wend
print "End of Record Set Reached"
wait
REM Selecciona otra tabla.
print CRT$ ("CLEAR");
goto tables
```

It has been another example within the library WIN32 our BBS as odbc.zip. This file contains a program that uses ODBC special format designed to illustrate exactly how to read and write a file for which you have the ODBC drivers.
All examples were updated programs in library WIN • "as demoprj.zip.

Chapter 11 CALLS FROM WIN32 API CET W32.

API is one of those acronyms that are used mainly to intimidate people. But API is simply a set of functions available for use by an application programmer.
Therefore, the Windows API is a set of functions that are part of Windows and are available for any Windows application. When you consider that Windows APIs have thousands of functions becomes aware of the large capacity.
It was mainly between 1991 and 1995 when the number of these functions increased since Microsoft added new dynamic link library (DLL) that extended the capacity WINDOWS operating system.
From version w/32 9.31I of CET BASIC can be called by CALL statements used within the BASIC sources, new functions that make use of the full power of the Windows API. These new features are developed by OMEGA SYSTEMS, SA and provides developers through files. "obj" or through a library file "w32funcs.lib". How to compile programs that use CET calls to these functions is:

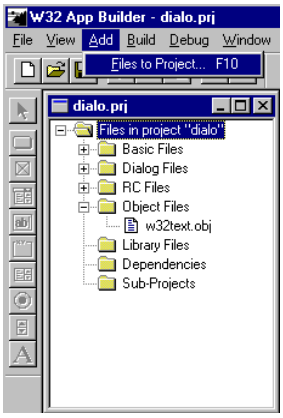
1) If you compile from the command line: <programa_basic.b> <programa_basic> Obw32-or <funcion1.obj> ... <funcionn.obj>

2) If the application is developed using W32 CET Application Builder, the files must be added. "Obj" Object Files within the group.

Or just add the file w32funcs.lib in group 'Library Files'.

We have developed several features to provide the opportunity to enhance the performance of applications W32 CET.

w32CreateCaret (width%, Hight%, addrof (retorno%)),

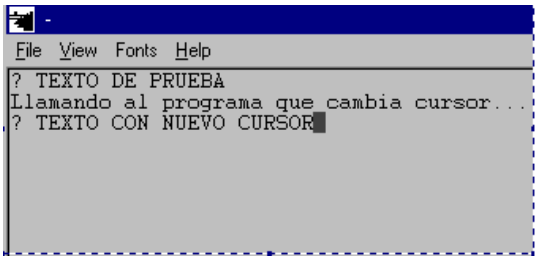


Lets you change the size of the cursor indicating the position of the text in any Windows box. The width parameter% is the sign of the cursor width in logical units. High% is the height of the cursor. The return value is non-zero% if the call was successfully executed.

Ejemplo: cc.b

w32ExitWindows (codigo%, addrof (retorno%)).

Valid values for code% argument are:



0 if you want to restart the session WINDOWS

1 if you want to turn off the computer
2 if you want to restart your computer
5 if you want to turn off the computer without prompting the user for any open programs
6 if you want to restart your computer without prompting the user for any open programs

Return% will have a value other than 0 if executed properly.

The line of code that would shutdown: Call w32ExitWindows (1,addrof (retorno%))

Ejemplo: ew.b

w32PlaySound (fichero$,addrof (retorno%))

Can be used to play any audio file.
$ File is the audio file name with full path.
Return%, equal to 0 if executed properly.

Ejemplo: ps.b

w32MessageBox (texto$,titulo$,estilo%,addrof (retorno%))

Displays a message in a dialog box, waits for the user to click a button, and returns an Integer corresponding to the button chosen by the user.

The parameter $ text is the content of the dialog box. If consists of more than one line, you can separate them using a carriage return character (Chr $ (13)) or a line feed character (Chr $ (10)).

$ Title is a string expression displayed in the title bar of the dialog box, if assigned to "" the title is "Failed".

Style%. Numeric expression that is the sum of values specifying the number and type of buttons that are intended to display, the icon style to use and the identity of the default button.

The possible values are:

0 Display OK button only.
1 Display OK and Cancel buttons.
2 Displays the Abort, Retry, and Ignore.
3 shows the buttons Yes, No, and Cancel.
4 Display Yes and No buttons
5 Display Retry and Cancel buttons.

16 shows the critical message icon.
32 shows the warning icon question.
48 shows the warning icon.
64 shows the information message icon.

0 First button is default.
256 Second button is default.
512 Third button is default.

0 Application modal, the user must respond to the message box before continuing work in the current application.
4096 modal system, all applications are suspended until the user responds to the message box.

The first group of values (0-5) describes the number and type of buttons displayed in the dialog box and the second group (16, 32, 48, 64) describes the icon style, the third group (0, 256, 512) determines the default button, and the fourth group (0, 4096) determines the modality of the message box. When adding numbers to get the final value of the style argument% uses only one number from each group.

The return value%, can be one of the following values indicating the button pressed by the user:

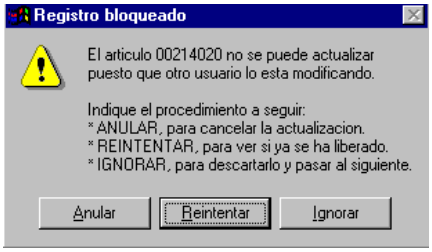
1 OK

2 Cancel
3 Cancel
4 Retry
5 Ignore
6 Yes
7 No

The following example illustrates the potential program dialog that could be presented to the user in the event that one of the records in a sequence to be updated is locked by another user.

```
call cForceExit
cAppTitle call ("My Application")

arti $ = "00214020"
$ title = "Record locked"
$ line1 = "Article" arti & $ & "can not be updated"
$ line2 = "because another user is modifying."
Line3 $ = "Specify the procedure:"
Line4 $ = "* CANCEL to cancel the upgrade."
linea5 $ = "* RETRY, to see if it has already been freed."
linea6 $ = "* ignore, to dismiss it and move to the next."

text $ = line1 $ & chr $ (10) & line2 $ & chr $ (10) & Chr $ (10) & Line3 $ & chr $ (10) &
Line4 $ & chr $ (10) & linea5 $ & chr $ (10) & linea6 $
w32MessageBox call ($ text, $ title, 4402, addrof (retcode%))
```

The execution result is the following:


Registro bloqueado
El articulo 00214020 no se puede actualizar
puesto que otro usuario lo esta modificando.
Indique el procedimiento a seguir:
* ANULAR, para cancelar la actualizacion.
* REINTENTAR, para ver si ya se ha liberado.
* IGNORAR, para descartarlo y pasar al siguiente.
Anular
Reintentar
Ignorar

The number 4402 is obtained from the sum of 2 (Abort, Retry, and Ignore), 48 (exclamation icon), 256 (the second button is the default) and 4096 (Dialogue supersedes any system window).

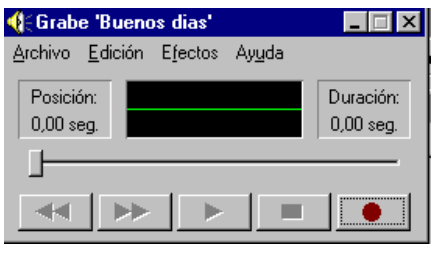
Example: mb.b

w32SetWindowText ($ window, $ title, addrof (return%))

The system searches the last open window whose title is equal to $ window and assigns that window title $ title.

Return% is equal to 0 if the title change is successful.
By way of example:

```
call cForceExit
Rem |Abrir la grabadora de sonidos
call cWinExec ("sndrec32",1)

Rem |Buscar en el sistema la ultima ventana abierta cuyo Rem |titulo contenga la expresion
'Grabadora de sonidos' y Rem |asignar a esa ventana el titulo "Grabe 'Buenos dias'"
call w32SetWindowText ("Grabadora de sonidos","Grabe 'Buenos dias'",addrof (retcode%))
wait
```



Ejemplo: swt.b

w32GetWindowSessions (ventana$,addrof (numero%)),

% returns the number of open windows in the system whose title is equal to the expression window $. This feature is useful for controlling the fact that the same user can open multiple instances of an application module.

The following example uses the function recursively w32GetWindowSessions to control that only open 5 sessions of a program.

```
cAppTitle call ("My Window")
w32GetWindowSessions call ("My Window", addrof (number%))
print number%
if number% <5
then
cWinExec call (". \ gws", 1)
IfEnd
wait
```

Example: gws.b

w32LBoxResetContent (),

Erases the contents of a list box control. Previously the focus should be placed on the list control whose contents are deleted. (Use this function to cSetFocus).

Example: lbrc.prj

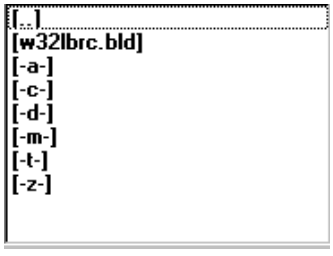
w32LBoxDir ($ directories, addrof (return%)),

Fill a list control with a list of directory names and units. Previously the focus should be placed on the list control desired. (Use this function to cSetFocus).

The parameter $ directory should contain a string that defines a DOS file specification. You can include extended search characters "?" And "*".

Directory names appear in the list enclosed in brackets ([])
The units are listed on the form [-x-], where x is the drive letter.

The parameter returns% returns the entry number, starting from 0, the last line that was added to the list. Zero on failure.



Ejemplo: lbrc.prj

w32BloadImage (fichero$,ancho%,alto%,addrof (retorno%))

Allocate an image. bmp, whose full path is specified in the parameter file $, to a command button that currently has focus. The image is set to the specified width and height.

Example: lbrc.prj

w32BFLoadImage ($ file, width%, height%, addrof (return%))

Has the same behavior as the previous function, but also causes the particular button is flat thus illustrative images that can be placed in the dialogues.

Ejemplo: lbrc.prj

w32ClipCopyData (text$,addrof (retcode%)).
This function places the string $ text to the clipboard. If executed successfully, retcode% will have a nonzero value.

w32ClipIsAvailable (addrof (result%)).
This function determines whether the clipboard contains data in text format.
Result%: A nonzero value indicates that data is available.

w32ClipPasteData (addrof ($ text), addrof (retcode%)).
This function retrieves data from the clipboard and assigned to $ text. If executed successfully, retcode% will have a nonzero value.

w32CopyFile ($ source, $ target, addrof (error $), addrof (retcode%)).
This function copies an existing file to one destination. If the target file already exists, the function overwrites it and returns a successful return value, this is a non-zero value. If the function returns with an error condition, the description thereof will be returned in the variable $ Error

w32CopyFile call ("c: \ windows \ w32app.w32", "\ \ omega \ public \ w32app.w32" addrof (error $), addrof (return%))

w32CreateDirectory ($ dir, addrof (retcode%)).
This function creates a new directory.
Retcode%: Zero indicates error.

w32EditUndo ().
Undoes the last operation of an edit control. The focus must be on the edit control. (Use cSetFocus function for this purpose).

w32FileSize (file $, addrof (size%) addrof (retcode%)).
Returns the size in bytes of the file specified.
Retcode%: Zero indicates error.

w32FileTime ($ file, addrof (year%), addrof (month%), addrof (day%), addrof (time%), addrof (min%), addrof (s%), addrof (retcode%)).
Returns the time of last modification of the file.
Retcode%: Zero indicates error.

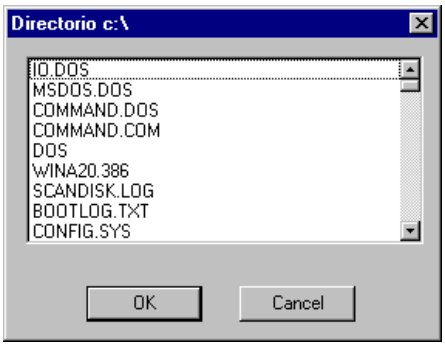
w32FindFirstFile ($ pattern, addrof (firstfile $), addrof (handle%)).
W32FindFirstFile function opens a search handle% and returns the name of the first file whose name matches the specified $ pattern. Once the handle is returned% search, use the search function to successive w32FindNextFile files matching the pattern $.

w32FindNextFile (handle% addrof (nextfile $), addrof (retcode%)).
This function continues the search for files from a previous call to the function w32FindFirstFile.

Handle%: search handler returned by a previous call to w32FindFirstFile
$ NextFile name file or directory in sequence found
Retcode%: Zero may indicate that an error has occurred, or that there are no more files.

```
dim matriz$ (500)
indice=1
call w32FindFirstFile ("c:\*.*",addrof (fichero$),addrof (handle%))
if handle%<>0
then
        matriz$ (indice)=fichero$
        call w32FindNextFile (handle%,addrof (fichero$),addrof (retorno%))
```

```
        while retorno%<>0
                indice=indice+1
                matriz$ (indice)=fichero$
                call w32FindNextFile (handle%,addrof (fichero$),addrof (retorno%))
        wend
ifend
call cChoiceListNS ("Directorio c:\",mataddrof (matriz$),addrof (sel$))
```



w32GetComputerName (addrof (name$)).

Returns computer name.

w32GetDiskFreeSpace (unidad$,addrof (bytesfree%),addrof (retcode%).

Returns the number of bytes that are free on the drive that was specified.
Retcode%: Cero indica error

w32GetWindowsDirectory (addrof (path$))

Returns the WINDOWS directory path.

w32GetDriveType (unidad$,addrof (retcode%)).

Returns information about the type of drive unit $.
Retcode% can be:

```
0       ERROR
1       LA UNIDAD NO EXISTE, O NO HAY NINGUN MEDIO INSERTADO
2       UNIDAD REMOVIBLE
3       UNIDAD FIJA
4       UNIDAD REMOTA
5       UNIDAD CDROM
6       DISCO RAM


call cOpenBrowser (addrof (nombre$),"Imagenes para Botones  (*.bmp)|*.bmp||")
if sch (0,nombre$,"\")<>0
then
        for i=1 to len (nombre$)
                if nombre$[i:i]="\" then break
        next i
        call w32GetDriveType (nombre$[1:i],addrof (retorno%))
        SELECT retorno%
        CASE 2
                Msgbox "No se permiten las unidades removibles"
        CASE 3
                rem call cSetFocus...
                rem call w32BLoadImage (nombre$,addrof (valor%))
        CASE 4
                Msgbox "No se permiten las unidades de red"
        CASE 5
                Msgbox "No se permiten las unidades de CD-ROM"
        CASE 6
                Msgbox "No se permiten los discos RAM"

        CEND
ifend
```

w32keybd_event (char$,char%).

This function can be used to enter a character in the input buffer as if it had been entered by the operator.
$ Char: The character you want to enter.
Char%: The character's ASCII code

w32Skeybd_event_down (key%).

This function can be used to simulate a keypress.
Key%: One of the following values:

```
BACKSPACE       8
TAB             9
RETURN  13
SHIFT           16
CONTROL 17
ALT             18
PAUSA           19
BLOQ. MAY.      20
ESCAPE  27
ESPACIO 32
AV. PAG.        33
RE. PAG.        34
FIN             35
INICIO  36
CUR. IZQ.       37
CUR. ARR.       38
CUR. DER.       39
CUR. ABJ.       40
INSERT  45
SUPR.           46
'0' a '9'  (48 - 57)  (Identicos a los valores ASCII)
'A' a 'Z'  (65 - 90)  (Identicos a los valores ASCII)
Teclas de función F1 a F12  (112 - 123)
```

w32Skeybd_event_up (key%).

This function can be used to simulate the release of a key.

w32MoveWindow (titulo$,x%,y%,width%,height%,addrof (retcode%)).

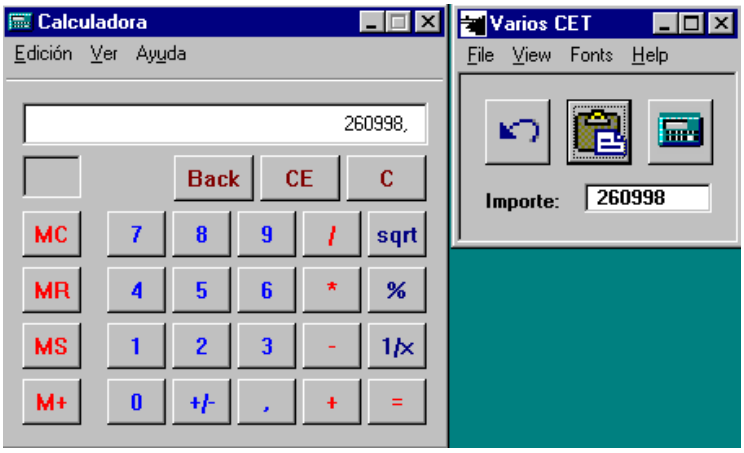
Change the position and size of the last open window whose title equals $ title
Retcode%: Zero indicates error

w32RemoveDirectory (dir$,addrof (retcode%)).

This function deletes the specified directory. The directory must be empty.
Retcode%: Zero indicates error

w32SetForegroundWindow (titulo$,addrof (retocode%)).

Active window set as the last open window whose title is equal to $ title parameter.
Retcode%: Zero indicates error.
Below is a snippet cal.prj program that illustrates the use of several functions documented above.




```
$SUB dlg1BtnClick
REM     Button Click save area
REM@# btnstart
SELECT CMDID%
```

```
CASE 202
        Rem Undo- Boton Deshacer
        call cSetFocus (dlg1HDLG%,100)
        call w32EditUndo ()
CASE 205
        Rem Paste- Boton Pegar
        call w32GetWindowSessions ("Calculadora",addrof (sesiones%))
        if sesiones%>0
        then
        call w32SetForegroundWindow ("Calculadora",addrof (retorno%))
        Rem - Simular CONTROL+C
        call w32Skeybd_event_down (17)
        call w32keybd_event ("C",asc ("C"))
        call w32Skeybd_event_up (17)
        call cAppWindowTop
        call w32ClipIsAvailable (addrof (ava%))
        if ava%<>0
        then
                call w32ClipPasteData (addrof (text$),addrof (retcode%))
                call cSetCtrlValue (dlg1HDLG%,100,text$)
        ifend
        ifend
CASE 206
        call w32GetWindowSessions ("Calculadora",addrof (sesiones%))
        if sesiones%=0
        then
                call cWinExec ("calc.exe",1)
        ifend
        call cAppWindowTop
        call w32MoveWindow ("Calculadora",0,0,276,272,addrof (retorno%))
        call w32SetForegroundWindow ("Calculadora",addrof (retorno%))
        Rem -Ver la calculadora Estandard
        call w32Skeybd_event_down (18) \rem Pulsar ALT
        call w32keybd_event ("V",asc ("V"))
        call w32Skeybd_event_up (18) \rem Liberar ALT
        call w32keybd_event ("E",asc ("E"))

CEND
REM@# btnend
GOSUB dlg1.BTNEND
$EXIT
```

call w32BMultiline ( )

Allows specifying multiple lines of text in a button. (The focus should be located previously on the same).

w32B3D call ()

Displays a button in its normal state of 3 Dimensions. (The focus should be located previously on the same).

w32BFlat call ()

Displays a button in its pressed state. (The focus should be located previously on the same).

w32GetDiskSpace call ($ unit, addrof (area%), addrof (ret%))

Returns the total drive capacity in bytes. Returns 0 on error.

w32SetFont call ($ source, points%, position% bold% italic% sub% strikeout%)

Sets the font of any control in which the focus is located. The orientation parameter designates the degrees of inclination of the text. The last 4 parameters take value 1 to enable and 0 to disable attribute.

w32CreateProgress call (x%, y%, width%, height%,% continuous, vertical% addrof (hand%))

Create a progress bar in the window that has focus. Continuous parameters and vertical%% can take values 1 or 0 to enable or disable the corresponding feature. The return value is necessary to refer to the bar in the rest of the functions that operate on the same.

w32ProgressSetRange call (hand% min% max%)

Sets the maximum and minimum values for the progress bar and redraws hand% to reflect the new range. (By default when creating the bar minimum value is 0 and the maximum is 100).

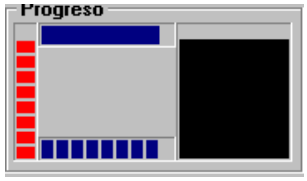
w32ProgressSetStep call (hand% step%)

Sets the step increment for a progress bar. The step increment is the amount by which the progress bar increases its current position when calling w32ProgressStepIt function. (Default to create the step sweep provides increased to 10).

w32ProgressStepIt call (hand%)

Advances the current position of the progress bar in a step increment and redraws the bar to reflect the new position. When the position exceeds the maximum range this function resets the current position to the progress indicator start again from the beginning.

w32ProgressSetColor call (hand% r% v%, y%)
Sets the color of the progress bar control.
r% - value of R (red) color; Valid values are 0 .. 255
v% - component value V (green) color; Valid values are 0 .. 255
a% - value component (blue) color; Valid values are 0 .. 255



Ejemplo: Create.prj

call w32CreateCalendar (x%,y%,ancho%,alto%,addrof (cal%))

Create a calendar control inside the window that has focus
The return value is necessary to refer to the timing in the rest of the functions that operate on the same

Ejemplo: Create.prj

call w32CalendarGetSel (cal%,addrof (año%),addrof (mes%),addrof (dia%))

Retrieves the currently selected date in the calendar cal%.

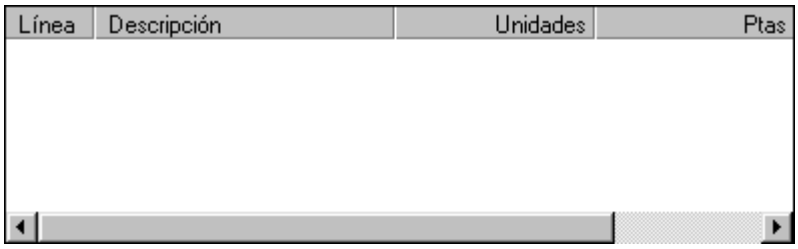
call w32CreateList (x%,y%,ancho%,alto%,addrof (lis%))

Create a list report within the window that has focus.
The return value is necessary to refer to the list in the rest of the functions that operate on the same.

call w32ListInsertColumn (lis%,titulo$,ancho%,alinea%,posi%,addrof (ret%))

Inserts a column in the list lis%.
$ Title-title associated with the column that appears in the header of the list.
Width% - column width in pixels.
Aligns% - Alignment of text entered in that column:
0. - Left
1. - Center
2. - Right.
Posi% - Index of the new column. (Starting at 0).
Ret% - Returns the index of the new column or -1 on error

Ejemplo: *Create.prj*



call w32ListInsertItem (lis%,texto$,item%,addrof (resu%))

Inserts a new item in the list lis%.
Text%. - Text element associated with the field (column) 0.
Item%. - The index of the new item. (Starting at 0).
Resu%. - Returns the index of the new item, -1 on error.

w32ListSetText call (lis%, $ text, item% column%, addrof (resu%))

Sets the text value of a given sub-item of the list lis%.
Item%. - Index of the item in the list.
Column%. - Index of the column, which defines the field belonging to the item% item whose text you want to set.
The resu% value is nonzero if the call has been executed successfully.

w32ListGetText call (lis% item% column%, addrof ($ text), addrof (ret%))

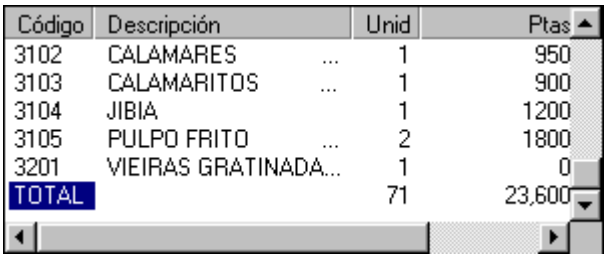
Retrieves the value of the text, text $, for a given sub-item of the list lis%.
Item%. - Index of the item in the list.
Column%. - Index of the column, which defines the field belonging to the item% item whose text you want to retrieve.
Ret% value returns the retrieved string length.

w32ListGetSelItem call (lis% addrof (item%))

Retrieves the index of the list element lis% selected by the user. On error -1 is returned.

call w32ListSetSelItem (lis%,item%,addrof (ret%))

Sets the item% item lis% list as the selected item of the same. Ret% value is nonzero if the call was successfully executed.



call w32ListGetItemCount (lis%,addrof (cuantos%))
Retrieves the number of items in the list lis%.

w32ListFindItem call (lis%, $ key, addrof (where%))
Retrieves the index of the list item lis% the value of the field (sub-item) 0 is equal to $ key. The value -1 is returned if the search is not successful.

w32ListSetColor call (lis% r% v%, y%)
Sets the text color of the list items lis%.
r% - value of R (red) color; Valid values are 0 .. 255
v% - component value V (green) color; Valid values are 0 .. 255
a% - value component (blue) color; Valid values are 0 .. 255

w32ListDeleteItem call (lis% item% addrof (ret%))
Removes the item at index in the list item% lis%.
Ret% value is nonzero if the call has been executed successfully.

w32ListDeleteAllItems call (lis% addrof (ret%))
Removes all items from the list lis%.
Ret% value is nonzero if the call has been executed successfully.

w32ListDeleteColumn call (lis%, column% addrof (ret%))
Removes the column column% of the list lis%
Ret% value is nonzero if the call has been executed successfully.

all w32SetFocus (hand% addrof (ret%))

Sets the control referenced by hand% as the new control that has the input focus mouse and keyboard.
The return value ret% contain the reference to the control that previously had the focus.
w32CreateAnimation call (x%, y%, width%, height%, addrof (animal%))

Create animated control within the window that has focus, ani% value will be used in reference to the control.

w32AnimationOpen call (ani%, $ file, addrof (return%))
Opens a file $. Avi (specify complete path) is associated to the control ani% and displays in its initial state. Return% is 0 on error.

call w32AnimationPlay (ani%,addrof (retorno%))
Sign uninterrupted playback of avi file associated with the control ani%. This reproduction is performed in the background while the remainder of the application is running. Return% is 0 on error.

Ejemplo: Create.prj

call w32AnimationStop (ani%,addrof (retorno%))
Stop playing avi file associated with the control ani%. Return is 0 on error

w32MAPISendDocuments call ($ paths, $ file, addrof (ret%))
Send an email message specifying a list of attachments, and shows the user a dialog box to enter the recipient's address.

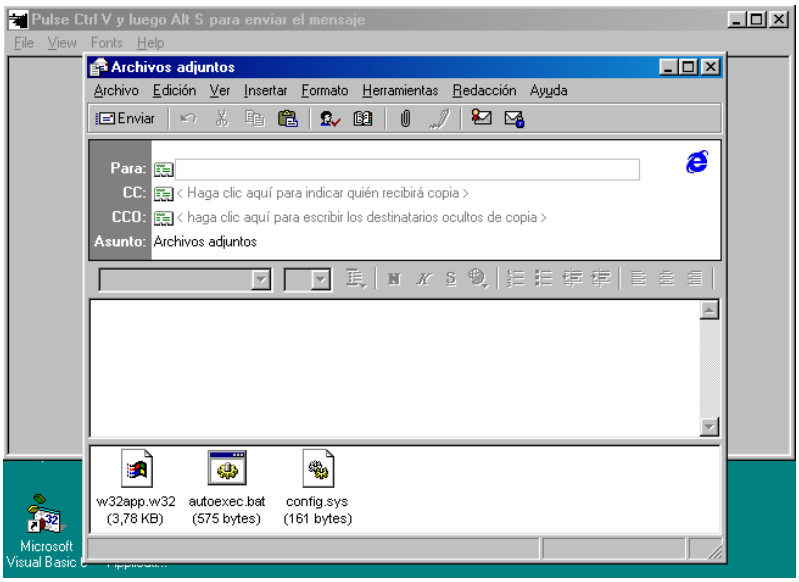
The $ path parameter must specify the full path of the file you wish to attach separated by semicolons. In parameter $ file must be specified, separated by semicolons, the name Asinara each of the attachments to the destination.

example:

```
call w32MAPISendDocuments
("c:\windows\w32app.w32;C:\autoexec.bat;c:\config.sys","w32app.w32;autoexec.bat;config.sys"
,addrof (ret%))
```

call w32MAPIReadMail (dirdestino$,remitente$,addrof (files%),addrof (mes%))

Processes all messages that are in the inbox of the mail client system sender whose address is contained in the parameter $ sender, copy the attached files to the directory $ dirdestino and finally delete the messages.



The parameter files% return the number of files that have been copied.
The parameter month% return the number of messages processed. O -1 on error.

example
```
call w32MAPIReadMail ("c:\correo\","@activanet.es",addrof (files%),addrof (ret%))
```

call w32DestroyWindow (handle%)

Deletes the control created by the function call w32Create
[Calendar|Progress|List|Animation|...]

Ejemplo:

```
print at$ (1,1);"Backup..."
call w32CreateAnimation (1,15,150,50,addrof (ani%))
call w32AnimationOpen (ani%,".\file.avi",addrof (ret%))
call w32AnimationPlay (ani%,addrof (ret%))
wait
call w32AnimationStop (ani%,addrof (ret%))
wait
call w32DestroyWindow (ani%)
print at$ (1,1);"End Backup"
wait
```

call w32ListEnsureVisible (lis%,item%,addrof (ret%))

Specifies that the line item from the list lis%% must be visible. This function will move the vertical scroll control if necessary.

w32ListGridLines call (lis% addrof (ret%))

Displays grid lines dividing the list lis%.

call w32ListClearGridLines (lis%,addrof (ret%))

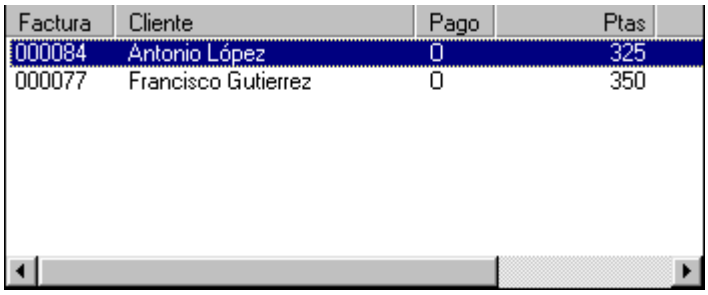
You void grid gridlines on the list lis%

w32CalendarSetSel call (cal% year%, month%, day%)

Sets the selection calendar created with lime% w32CreateCalendar call. The year value must be a 4 digit number.

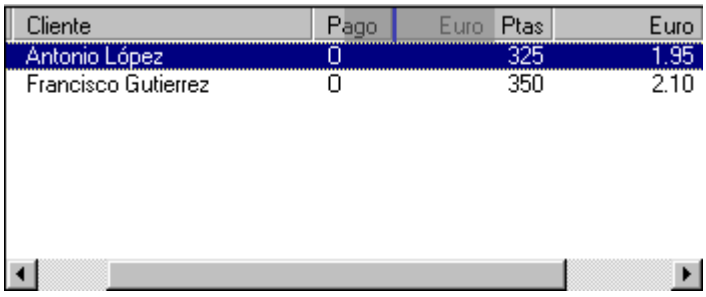
w32ListFullRowSel call (lis% addrof (return%))

It enables when you select any line in the list lis% all the same fields are marked. If you want to set this property to the list should call this function only once after the creation of the list with w32CreateList.



call w32ListDragDrop (lis%,addrof (retorno%))

Allows the user to drag and change the order of columns in a list control, without changing the internal reference to them from within the program. If you want to set this property to the list should call this function only once after the creation of the list with w32CreateList.



call w32CreateTip (texto$,addrof (tip%))

This function can be used to display help text associated Multiline any control dialogues W32 CET.

$ Displays help text that is specified as a parameter, and returns a reference to the control that should be used in subsequent function calls w32TipUpdateText. Previously the focus should be placed on the control that you want to associate the help message. (Use call cSetFocus)

call w32TipUpdateText (tip%,nuevotexto$,addrof (retorno%))

Update the help text associated with the control tip%. Previously the focus should be placed on the control that you want to update the help text.

w32GetParent call (addrof (parent%))

Returns a reference to the parent window of the control that has focus. (Use cSetFocus) The typical use of this function is to obtain a reference to the dialogue when the focus is on one of its controls.

w32ResizeDialog (father%, x%, y%, width%, height%, addrof (return%))

Change the position and size of main window dialog (dialog box).
The first parameter must be the value returned by the function call w32GetParent. The following four parameters must be specified in pixels and indicate respectively the horizontal and vertical position of the left corner and the width and height of the dialog box.

w32ListSetBkColor call (lis% r% v%, y%)

Sets the background color of the text of the list items lis%.
r% - value of R (red) color; Valid values are 0 .. 255
v% - component value V (green) color; Valid values are 0 .. 255
a% - value component (blue) color; Valid values are 0 .. 255

w32EnableWindow call (hand%, status%)

Active (% = 1) or off (state% = 0) control.
The first parameter should be the reference value obtained for the control function call w32Create [Calendar | Progress | List | Animation | ...]

When a control is disabled grayed out and can not get the input focus.

w32FtpGet call (FTPSite $, $ user, $ key, $ ficheroftp, ficherolocal $, addrof (return%))

Gets a file from an ftp site and saves it locally. Cetftp dll is needed.

The return value can be:

1 Operation completed successfully.
-1 Error in the connection to the ftp site
-2 Error processing file.

Example:

```
call w32FtpGet (Return%))
```

w32FtpPut call (FTPSite $, $ user, $ key, $ ficheroftp, ficherolocal $, addrof (return%))

Send a file to an FTP location. Cetftp dll is needed.

The return value can be:

1 Operation completed successfully.
-1 Error in the connection to the ftp site
-2 Error processing file.

Call w32PrinterDlg (addrof (context%), addrof ($ printer), addrof (or%), addrof (c%))

Presents a dialog box for the user to select a printer, print settings and define the orientation and number of copies. Requires cetprint.dll dll
All values of this function are to return, and have the following interpretation:

%-Context, reference to the object printer. It will serve as an input parameter in other functions that return properties of the printed page.
-$ Printer, printer name chosen by the user.
-O%, page orientation specified by the user (1-Vertical, 2-Horizontal)
-C%, number of copies specified by the user.

Example: prt.b

w32PrinterContext call ($ printer, addrof (context%))

Create a reference to the context of the specified printer by printer $. Requires cetprint.dll dll.

Example: prt.b

w32PrinterDeleteContext call (context%)

Destroy the reference to the context of the printer. It is intended to free up resources. Requires cetprint.dll dll.

Example: prt.b

w32PrinterOffsetX call (context% addrof (bordeizq%))

Returns in pixels of the left edge. unprintable page printer context referenced by%. Requires cetprint.dll dll.

Example: prt.b

w32PrinterOffsetY call (context% addrof (bordesup%))

Returns the value of pixels of the upper edge of the printable page printer not referenced by context%. Requires cetprint.dll dll.

Example: prt.b

w32PrinterWidth call (context% addrof (anchuratotal%))

Returns the width in pixels of the total page (printable area + unprintable area) printer context referenced by%. Requires cetprint.dll dll

Example: prt.b

w32PrinterHeight call (context% addrof (alturatotal%))

Returns the height in pixels of the total page (printable area + unprintable area) printer context referenced by%. Requires cetprint.dll dll

Example: prt.b

w32ListSortItems call (lis%, column% alphanumeric% order% addrof (ret%))

To sort the list items according to the values and types of the fields in the specified column.

lis%, reference to the list returned by the function w32CreateList
Column% Column to sort by list. (Starting at 0)
alphanumeric%, 0 alphanumeric sort, 2 and 3 numeric sort sort by date.
order%, 1 up and 2 down.

w32CreateUpDown call (ini% end% addrof (updown%))

Associates an interval to control existing edit control.

(The focus should be placed earlier in the edit box. Use for this
cSetFocus function)

The order parameters and ini%% determine the numerical range of advance.

Example:
Be associated Up-Down control to an edit box with ID 100 in the dialogue dlg2HDLG%. It is intended to specify a range of 1 to 10 advance. The code would be:

cSetFocus call (dlg2HDLG%, 100)
w32CreateUpDown call (1.10, addrof (updown%))

w32ListCheckBoxes call (lis% addrof (return%))

Add check boxes to each row of the list lis%. If you want to set this property to the list should call this function only once after the creation of the list with w32CreateList.

Nombre	Descripción	Modificado	Tamaño	Registros
☑ FAMI	FAMILIA ARTICULOS	27/06/2000 19:08	2 KB	35
☐ LaPR	LINEA ALBARAN PR...	28/06/2000 12:12	2 KB	7
☐ LPPR	LINEA PEDIDO PRO...	27/06/2000 19:08	2 KB	29
☑ LTIC	LINEA TICKETS	28/06/2000 12:19	26 KB	794
☑ MESA	MESAS	27/06/2000 18:54	5 KB	306
☐ PARA	PARAMETROS	28/06/2000 12:12	2 KB	9
☐ PART	PROVEEDOR/ARTI...	27/06/2000 19:08	2 KB	7

call w32ListGetCheck (lis%,fila%,addrof (sel%))

Check if the checkbox of a particular row% of the list is marked lis%.
Sel% value will be 0 if not checked and nonzero if it is.

w32ListSetCheck call (lis%, row%, status% addrof (return%))

This function sets the state mark the checkbox of a particular row% of the list lis%.
The status value% must be 0 to clear the check and 1 to activate.

call w32GetFont (addrof ($ source), addrof (% points), addrof (inclination%), addrof (bold%), addrof (italics%), addrof (underlined%), addrof (strikethrough%))

W32GetFont function returns the attributes of the source that has the control that has focus. (Use to call cSetFocus)
The meaning of the parameters is identical to the function w32SetFont.

w32GetWindow call (father% ID%, addrof (hand%))

Returns a reference to a control in a dialog box. The first parameter should be returned by the function call w32GetParent. The second parameter must be the ID value of the control you want to get the reference.

w32SetFontH call (hand%, $ source, points%, position% bold% italic% sub% strikeout%)

Sets the font control% referenced by hand. The orientation parameter designates the degrees of inclination of the text. The last 4 parameters take value 1 to enable and 0 to disable attribute.

Ejemplo:

Se desea asignar el tipo de fuente Tahoma de 12 puntos al control de edición con ID 100 perteneciente al diálogo dlg1.

```
$SUB dlg1Init
REM     Initialize dialog save area
REM@# inistart
Rem poner el foco en un boton cualquiera  (203 p.ej) para hallar la referencia al cuadro de
dialogo.
Call cSetFocus (dlg1HDLG%,203)

Rem Obtener la referencia al dialogo. Solo es necesario llamar a esta funcion una vez al
inicio del dialogo.
call w32GetParent (addrof (padre%))

Rem Obtener la referencia al control 100 del dialogo dlg1
Call w32GetWindow (padre%,100,addrof (ed100%))

Rem Cambiar la fuente
Call w32SetFontH (ed100%,”Tahoma”,13,0,0,0,0,0)

Rem En cualquier otra seccion del dialogo se podra cambiar la fuente o cualquier otro
atributo del control simplemente especificando ed100% como referencia del mismo en la
funcion deseada  (w32ResizeWindowH, por ejemplo)
REM@# iniend
```

w32ResizeWindowH (hand%,x%,y%,ancho%,alto%,addrof (retorno%))

Change the position and size of the control referenced by hand%.
The following four parameters must be specified in pixels and indicate respectively the horizontal and vertical position of the left corner and the width and height that are assigned to the control.

w32CreateImageH call (hand%, file $, x%, y%, width%, height%, addrof (ret%))
Draws an image. Bmp, whose full path is specified in the parameter file $, in the control referenced by hand%.

Example: Show the image stored in the file at position logo.bmp 15.10 DLG1 dialogue with 270 pixels wide and 135 high pixles

```
$SUB dlg1Init
REM     Initialize dialog save area
REM@# inistart
Rem poner el foco en un boton cualquiera  (200 p.ej) para hallar la referencia al cuadro de
dialogo.
Call cSetFocus (dlg1HDLG%,200)

Rem Obtener la referencia al dialogo. Solo es necesario llamar a esta funcion una vez al
inicio del dialogo.
call w32GetParent (addrof (padre%))

...

call w32CreateImageH (padre%,"logo.bmp",15,10,270,135,addrof (ret%))

REM@# iniend
```

call w32EditENTERn (hand%,IDbotonESC%)

Where n is a value from 1 to 9. Calling this function group allows the user to validate the information entered in any edit box
a dialogue by pressing the ENTER key, so that the focus goes to
next control. This special behavior only affects the editing controls to
which they are found associated function.

The interpretation of the parameters is as follows:

Hand%, referring to the edit control.
IDbotonESC%, a value between 200 and 299 that designates the ID button number, real or fictional, whose associated section in $ SUB BtnClick subroutine code is executed when the user presses ESCAPE being located in the control edition.

The usual place where to place the call to this function is the end of the section $ SUB Init.

In all dialogues that make up an executable to use a different function name for each edit control. So, for this special processing associated edit controls three functions to be used ...

Call w32EditENTER1...
Call w32EditENTER2...
Call w32EditENTER3...

Ejemplo:

```
$SUB dlg1Init
REM     Initialize dialog save area
REM@# inistart
Rem poner el foco en un boton cualquiera  (200 p.ej) para hallar la referencia al cuadro de
dialogo.
Call cSetFocus (dlg1HDLG%,200)

Rem Obtener la referencia al dialogo. Solo es necesario llamar a esta funcion una vez al
inicio del dialogo.
call w32GetParent (addrof (padre%))
...

Rem Obtener la referencia al control de edición con ID 100
Call w32GetWindow (padre%,100,addrof (ed100%))
Rem Obtener la referencia al control de edición con ID 101
Call w32GetWindow (padre%,101,addrof (ed101%))

...

Rem Proceso de enter para id 100 y 101, al pulsar ESC se Rem
pulsa el boton 201 en ambos casos
call w32EditENTER1 (ed100%,201)
call w32EditENTER2 (ed101%,201)
REM@# iniend
```

call w32SetCtrlColorn (padre%,control%,rt%,vt%,at%,rf%,vf%,af%)

Where n is a value from 1 to 19. Calling this function group to define the text color and background of any edit control, List Box, Combo Box (Drop down list) and static controls (text, radio button, check box and group boxes).

The interpretation of the parameters is as follows:

Father%, reference to the dialog. (Value returned by the function w32GetParent)
control%, reference to control the colors are assigned. (Value returned by the function as 1st parameter w32GetWindow estableciento father% value, and as the second parameter control id)

rt% - value of R (red) text color; Valid values are 0 .. 255. Use the value -1 to indicate that you want to use the default color for that control.
vt% - component value V (green) text color; Valid values are 0 .. 255. If parameter rt% -1 value is specified, in this you can specify the value returned by w32ImageBackgroundn if you want the background color is the same as the tapestry of dialogue.
at% - value component (blue) text color; Valid values are 0 .. 255

rf% - value of R (red) the background color of the text; Valid values are 0 .. 255. Use the value -1 to indicate that you want to use a transparent background for that control.
vf% - component value V (green) background color of the text; Valid values are 0 .. 255
af% - value component (blue) background color of the text; Valid values are 0 .. 255

The usual place where to place the call to this function is the end of the section $ SUB Init.

In all dialogues that make up an executable should not be called more than once to the same function in this group. Thus, for defenir controls the color of three functions to be used ...

Call w32SetCtrlColor1 ...
Call w32SetCtrlColor2 ...
Call w32SetCtrlColor3 ...

Some of the color combinations for the most common:

```
0,0,0 black
Yellow 255,255,0
Dark Yellow 128,128,0
Red 255,0,0
Dark Red 128,0,0
Magenta 255,0,255
Dark Magenta 128,0,128
Blue 0,0,255
Dark Blue 0,0,128
Cyan 0255255
0128128 Dark Cyan
Green 0,255,0
Dark Green 0128128
Grey 192,192,192
Dark Grey 128128128
White 255 255 255
```

Example:
Be assigned the following definition of colors to various controls of a dialog

El fragmento de código a ejecutar sería:

```
$SUB dlg1Init
REM     Initialize dialog save area
REM@# inistart
Rem poner el foco en un boton cualquiera  (200 p.ej) para hallar la
rem referencia al cuadro de dialogo con la siguiente llamada a
rem w32GetParent.
Call cSetFocus (dlg1HDLG%,200)


Rem Obtener la referencia al dialogo. Solo es necesario llamar a esta funcion una vez al
inicio del dialogo.
call w32GetParent (addrof (padre%))
...

Rem Obtener la referencia a cada control indicando su ID
call w32GetWindow (padre%,701,addrof (c701%)) \rem Caja Combo
call w32GetWindow (padre%,504,addrof (c504%)) \rem Importe
call w32GetWindow (padre%,101,addrof (c101%)) \rem Año
call w32GetWindow (padre%,800,addrof (c800%)) \rem Facturas
call w32GetWindow (padre%,801,addrof (c801%)) \rem presupuestos
call w32GetWindow (padre%,900,addrof (c900%)) \rem Por fecha
call w32GetWindow (padre%,901,addrof (c901%)) \rem Por importe
call w32GetWindow (padre%,300,addrof (c300%)) \rem List Box

call w32SetCtrlColor1 (padre%,c701%,255,0,0,-1,0,0)
call w32SetCtrlColor2 (padre%,c504%,-1,0,0,255,255,0)
call w32SetCtrlColor3 (padre%,c101%,128,0,0,-1,0,0)
call w32SetCtrlColor4 (padre%,c800%,255,0,255,-1,0,0)
call w32SetCtrlColor5 (padre%,c801%,128,0,128,-1,0,0)
call w32SetCtrlColor6 (padre%,c900%,0,0,255,-1,0,0)
call w32SetCtrlColor7 (padre%,c901%,0,0,128,-1,0,0)
call w32SetCtrlColor8 (padre%,c300%,0,128,0,-1,0,0)

...
```

```
REM@# iniend
```

call w32CreateListH (padre%,x%,y%,ancho%,alto%,addrof (lis%))

This function has the same interpretation as w32CreateList. The difference is that you no longer have to put the focus on a control to create the list within it, but rather the x% and y% are relative to the overall dialog box referenced by the parent parameter% (use the call to w32GetParent for this first parameter).

All lists of desired process events (see w32ListGetEventsn) must have been created with the call to w32CreateListH.

w32ListGetEventsn call (father% lis% button% addrof (event%), addrof (param1%), addrof (param2%))

Where n is a value from 1 to 9. Calling this function group can point to the system controls whose list of events generated by the user want to be notified.
The feature allows you to specify a dummy button ID whose associated section is the code to execute when an event occurs on the list. The last three parameters indicate the names of the variables that will hold at each instant information on the type of event occurred and its properties.

The interpretation of the parameters is as follows:

Father%, referring to the dialog box. (The value returned by the call to w32GetParent).
lis% control identifier list. (The value returned by the call to w32CreateListH).
button%, a value between 200 and 299 that designates the ID button number, real or fictional, whose associated section in $ SUB BtnClick subroutine code is executed when events occur.
Event% variable name that will contain the value of the number of events produced at each instant.
Param1%, the first property of the event. It returns -1 if not applicable.
Param2%, the second property of the event. It returns -1 if not applicable.

The information available for each type of event is as follows:

Evento	*Descripción*	*Propiedad 1*	*Propiedad 2*
1	Click con el botón izq. del ratón	Item (fila) de la lista	Sub-item (columna)
2	Cambio de estado (Selección o Check Box)	Item (fila) de la lista	-1
3	Doble click con el botón izq. del ratón	Item (fila) de la lista	Sub-item (columna)
4	Click con el botón derecho del ratón	Item (fila) de la lista	Sub-item (columna)
5	Pulsación de una tecla	Valor (ver w32Skeybd_event_down)	-1
6	Click en el título de una columna	-1	Columna de la lista

The usual place where to place the call to this function is the end of the section $ SUB Init.

In all dialogues that make up an executable should not be called more than once to the same function in this group. Thus, to define the event processing list controls three functions to be used ...

Call w32ListGetEvents1...
Call w32ListGetEvents2...
Call w32ListGetEvents3...

Este grupo de funciones reemplaza al grupo de funciones w32SetEvents*n*

Ejemplo:

```
$SUB dlg1BtnClick
REM     Button Click save area
REM@# btnstart
SELECT CMDID%
...
CASE 250
```

```
        Rem Notificaciones para la lista lis%
        Rem evento%->Numero de evento
        Rem l_item%->Propiedad 1
        Rem l_sub%->Propiedad 2
        SELECT evento%
        CASE 1
                Rem Click en un campo
                rem Msgbox "Click en el campo: "&str$ (l_item%)&","&str$ (l_sub%)

        CASE 2
                Rem Cambio de seleccion, o bien cambio de casilla de verificacion
                rem Msgbox "Ha cambiado el item: "&str$ (l_item%)
                call w32ListGetSelItem (lis%,addrof (item%))
                if item%<>-1
                then
                call w32ListGetText (lis%,item%,0,addrof (text$),addrof (ret%))
                call cSetCtrlValue (dlg1HDLG%,100,text$)
                ifend

        CASE 3
                Rem Doble Click en un campo
                Msgbox "Doble Click en el campo: "&str$ (l_item%)&","&str$ (l_sub%)

        CASE 4
                Rem Click con el botón de derecho en un campo
                Msgbox "Click con botón derecho en el campo: "&str$ (l_item%)&","&str$
(l_sub%)

        CASE 5
                Rem Se ha pulsado la tecla:
                Msgbox "Tecla: "&str$ (l_item%)

        CASE 6

                Rem Click en la cabecera de la columna
                Msgbox "Click en columna: "&str$ (l_sub%)
                call w32ListSortItems (lis%,l_sub%,0,1,addrof (ret%))

        CEND

...
CEND
REM@# btnend
GOSUB dlg1.BTNEND
$EXIT



$SUB dlg1Init
REM     Initialize dialog save area
REM@# inistart

call w32GetParent (addrof (padre%))
...
call w32CreateListH (padre%,10,195,475,180,addrof (lis%))
...
call w32ListGetEvents1 (padre%,lis%,250,addrof (evento%),addrof (l_item%),addrof (l_sub%))
...
REM@# iniend
```

call w32EditENTER (indice%,hand%,IDboton%,addrof (tecla%))

The call to this function allows the user to validate the information entered in any edit box
a dialogue by pressing the ENTER key, so that the focus goes to
next control. This special behavior only affects the editing controls to
which they are found associated function.
Furthermore it can process any key pressed by the user.

The interpretation of the parameters is as follows:

index% integer value between 1 and 99
Hand%, referring to the edit control.
IDboton%, a value between 200 and 299 that designates the ID button number, real or fictional, whose associated section in $ SUB BtnClick subroutine code is executed when the user presses any key being located in the
edit control.

key% variable name that will contain the value of the key pressed in every moment. (For a full account of the values of the keys see w32skeybd_event function documentation)

The usual place where to place the call to this function is the end of the section $ SUB Init.

In all dialogues that make up an executable to use a different value for each index% edit control.
So, to associate this special processing three edit controls to use functions could be ...

Call w32EditENTER (1, ...
Call w32EditENTER (2, ...
Call w32EditENTER (3 ...

Note 1: This function replaces w32EditENTERn function group, although its use is not inconsistent set

Note 2: Although the most obvious use of this feature is to edit controls can also be used with list controls (List Box)

Ejemplo:

```
$SUB dlg1BtnClick
REM     Button Click save area
REM@# btnstart
SELECT CMDID%
...
CASE 230
       SELECT tecla100%
        CASE 112
                    Rem Tratar la pulsacion de la tecla F1

        CEND

CASE 231
    SELECT tecla101%
    CASE  33
            Rem Re Pág.
    CASE 34
            Rem Av Pág.
    CEND
CEND
REM@# btnend
GOSUB dlg1.BTNEND
$EXIT



$SUB dlg1Init
REM     Initialize dialog save area
REM@# inistart
Rem poner el foco en un boton cualquiera  (200 p.ej) para hallar la referencia al cuadro de
dialogo.
Call cSetFocus (dlg1HDLG%,200)

Rem Obtener la referencia al dialogo. Solo es necesario llamar a esta funcion una vez al
inicio del dialogo.
call w32GetParent (addrof (padre%))
...

Rem Obtener la referencia al control de edición con ID 100
Call w32GetWindow (padre%,100,addrof (ed100%))
Rem Obtener la referencia al control de edición con ID 101
Call w32GetWindow (padre%,101,addrof (ed101%))

...

Rem Proceso de enter para id 100 y 101, al pulsar tecla se ejecutar el código de los
botones ficticios 230 y 231
call w32EditENTER (1,ed100%,230,addrof (tecla100%))
call w32EditENTER (2,ed101%,231,addrof (tecla101%)

REM@# iniend
```

call w32BrowseForFolder (padre%,titulo$,addrof (dir$),addrof (ret%))

Presents the user with a dialog box to select a directory.
Father% is the reference to the active application window. Must be the value
returned by the call to w32GetParent.
$ title help text that appears in the dialog box
$ dir full directory path chosen by the user.
ret% 0 if the user pressed the cancel button
example:

call w32BrowseForFolder (padre%,"Seleccione el directorio destino para la copia de seguridad",addrof (texto$),addrof (ret%))

call w32FileCopyProgress (padre%,origen$,destino$,addrof (ret%))

Copy all the files specified by the string $ source $ destination directory dialog showing the progress of the copy.
The parent parameter% is the reference to the active application window. It must be the value returned by the call to w32GetParent.
If the extension of the files specified in the source parameter $ is *. * Will copy all the files and subdirectories complete structure.
The parameter ret% return a nonzero value if the user canceled the copy or if an error occurred.

w32FileDeleteProgress call (father%, $ source, addrof (ret%))

w32FileRenameProgress call (father%, $ source, $ destination, addrof (ret%))

w32FileMoveProgress call (father%, $ source, $ destination, addrof (ret%))

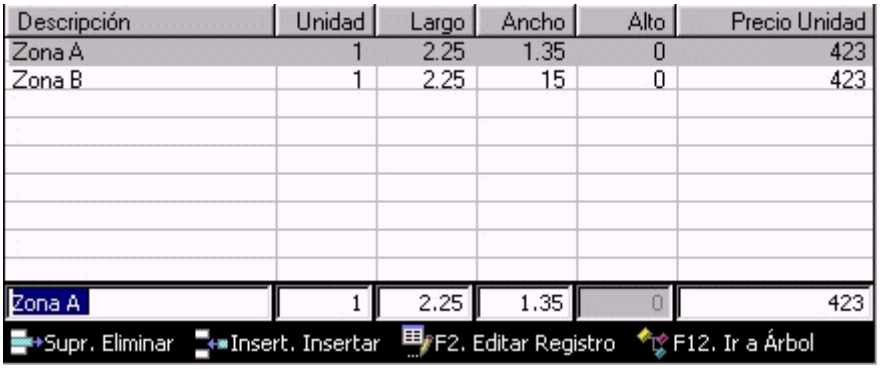
w32ListFindItemPartial call (lis%, $ string, position% addrof (where%))

Lets find rows of a list control from a specified position, whose field number 0 begins with the string specified as a parameter.
The syntax is:

lis% ID list control
$ string, text to be searched
Position%, row number from which the search is to be performed.
where%, index of the first row in which coincidence is found, or -1 if not successful searchpaths.

w32ControlFlat call (control%)

Paint a border around the control referenced by control% to give the appearance of two dimensions.
This could be the implementation of a grid in which the editing controls
they have activated w32ControlFlat.



call w32CreateImageList (ancho%,fichero$,addrof (ima%))

Create a reference to a file. Bmp must contain an image list.
The width of each individual image in pixels should be specified as the first parameter.
It returns a reference to the list of images in the last argument. the call
This function has no visual effect, only creates a reference to a list
of images that will then be referenced by its position in the list controls and
tree.

w32ListImageList call (lis%, ima% addrof (ret%))

Associated with the list lis% image list referenced by ima%. In most cases only have to make the call to this function only once per list.

w32ListSetImage call (lis% item% indice_ima% addrof (ret%))

Sets the image of the row item% belonging to the list lis% with the image of the list on lis% having índice_ima% index (starting from 0).

w32ListInsertItemEx call (lis%, text, item% indice_ima% addrof (resu%))

It has the same behavior as w32ListInsertItem function except that it is possible to specify a last parameter indicating the index of the image list associated with the control to be displayed in the row item%.
example:
You have created the following file image.bmp containing all possible images
to be displayed in rows in a list:

```
call w32CreateImageList (20,".\imagen.bmp",addrof (ima%)) \rem la anchura de cada icono es
de 20
call w32ListImageList (lis%,ima%,addrof (ret%)) \rem Asociar las imagenes a la lista
call w32ListSetImage (lis%,0,1,addrof (ret%)) \Establece el 2º icono como imagen del 1º
item
```

call w32CreateTreeH (padre%,x%,y%,ancho%,alto%,addrof (tree%))

Create a visual tree control.
The return value is necessary to refer to the tree in the rest of the functions that operate on the same.
The coordinates x% and y% are relative to the overall dialog box referenced by the parent parameter% (use w32GetParent call for this first parameter).

w32TreeInsertItem call (tree%, $ text, padre_en_arbol%, index% children% addrof (ref%))

Inserts a new element into the tree tree%, with text $ text

The parameter indicates the reference padre_en_arbol% element tree hierarchy immediately above the element to be inserted. When there is an ancestor in the tree above the padre_en_arbol% parameter value must be 0.

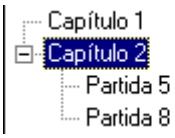
The index parameter% is an internal value that can be stored associated with that item in the tree.

The parameter indicates whether children% insert the item to be the ancestor of other lower-level elements. You must specify the value 1 if positive and 0 if not more branches will hang below the element to be inserted.

The ref parameter% returns a reference to the insert, which should be used to refer to the element in any function that requires it.

Example: The following lines of code produce the tree in Figure.

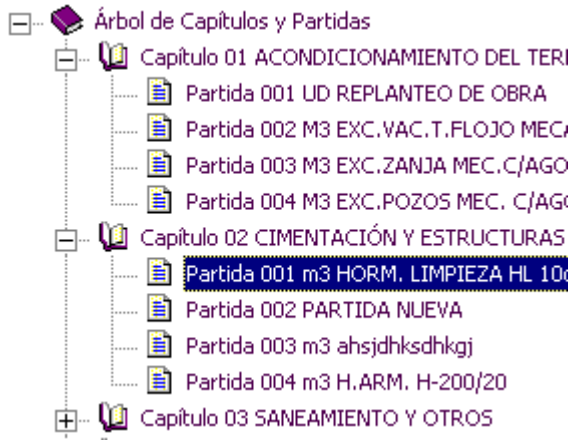
```
call w32TreeInsertItem (tree%,"Capítulo 1",0,0,0,addrof (cap1%))
call w32TreeInsertItem (tree%,"Capítulo 2",0,0,1,addrof (cap2%))
call w32TreeInsertItem (tree%,"Partida 5",cap2%,002001,0,addrof (ret%))
call w32TreeInsertItem (tree%,"Partida 8",cap2%,002002,0,addrof (ret%))
```



call w32TreeImageList (tree%,ima%,addrof (ret%))

Tree to tree% associated image list referenced by ima%.

w32TreeInsertItemEx call (tree%, $ text, padre_en_arbol%, index% children%, ima% addrof (ref%))
It has the same behavior as w32TreeInsertItem function except that it is possible to specify a last parameter indicating the index of the image list associated with the control that is displayed to the left of the item to insert.


Árbol de Capítulos y Partidas
Capítulo 01 ACONDICIONAMIENTO DEL TERI
Partida 001 UD REPLANTEO DE OBRA
Partida 002 M3 EXC.VAC.T.FLOJO MEC
Partida 003 M3 EXC.ZANJA MEC.C/AGO
Partida 004 M3 EXC.POZOS MEC. C/AG
Capítulo 02 CIMENTACIÓN Y ESTRUCTURAS
Partida 001 m3 HORM. LIMPIEZA HL 10
Partida 002 PARTIDA NUEVA
Partida 003 m3 ahsjdhksdhkgj
Partida 004 m3 H.ARM. H-200/20
Capítulo 03 SANEAMIENTO Y OTROS

call w32TreeExpand (tree%,ref%,addrof (ret%))

Expand the branch element referenced by ref% so that their children are visible.

w32TreeEnsureVisible call (tree% ref% addrof (ret%))

Specifies that the element tree tree ref%% must be visible. This function will move the vertical and horizontal scroll control if necessary.

w32TreeDeleteItem call (tree% ref% addrof (ret%))

Deletes the element referenced by ref% tree

w32TreeDeleteAllItems call (tree% addrof (ret%))
Clears all tree items tree%

w32TreeSetColor call (tree% r% v%, y%)
Sets the text color of tree tree items%.
r% - value of R (red) color; Valid values are 0 .. 255
v% - component value V (green) color; Valid values are 0 .. 255
a% - value component (blue) color; Valid values are 0 .. 255

w32TreeSetSel call (tree% ref% addrof (ret%))
Sets the element tree tree ref%% as the selected item of the same.

w32TreeGetSelItem call (tree% addrof (ref%))
Returns the reference of the currently selected item in the tree tree%.

w32TreeGetItem call (tree% ref% addrof ($ text), addrof (index%))
$ Text and returns the index% (internal value) belonging to the element referenced by ref%.

w32TreeGetParent call (tree% ref% addrof (padre_en_arbol%))
Returns a reference to the parent of the element referenced by ref%.

w32TreeGetChild call (tree% ref% addrof (son%))
Returns a reference to the first child of the element referenced by ref%.

w32TreeGetNextSibling call (tree% ref% addrof (brother%))
Returns a reference to the next sibling of the element referenced by ref%

w32TreeGetEventsn call (father% tree% button% addrof (etree%), addrof (ptree%))
Where n is a value from 1 to 9. Calling this function group can point to the tree controls system on which events generated by the user want to be notified.
The feature allows you to specify a dummy button ID whose associated section is the code to execute when an event occurs on the tree. The last two parameters indicate the names of the variables that will hold at each instant information on the type of event occurred and its property.

The interpretation of the parameters is as follows:

Father%, referring to the dialog box. (The value returned by the call to w32GetParent).
tree% control identifier tree. (The value returned by the call to w32CreateTreeH).
button%, a value between 200 and 299 that designates the ID button number, real or fictional, whose associated section in $ SUB BtnClick subroutine code is executed when events occur.
etree% variable name that will contain the value of the number of events produced at each instant.
ptree% ownership of the event.

The information available for each type of event is as follows:

Evento	*Descripción*	*Propiedad*
1	Click con el botón izq. del ratón	Referencia al elemento
2	Cambio de estado (Selección)	Referencia al elemento
3	Doble click con el botón izq. del ratón	Referencia al elemento
4	Click con el botón derecho del ratón	Referencia al elemento
5	Pulsación de una tecla	Valor (ver w32Skeybd_event_down)
6	Expansión de una rama	Referencia al elemento

The usual place where to place the call to this function is the end of the section $ SUB Init.

In all dialogues that make up an executable should not be called more than once to the same function in this group. Thus, to define the event processing tree controls three functions to be used ...

Call w32TreeGetEvents1 ...
Call w32TreeGetEvents2 ...
Call w32TreeGetEvents3 ...

w32DefineRegionn call (father% button% addrof (control%), addrof (ix%), addrof (i%), addrof (dx%), addrof (d%))

Where n is a value from 1 to 9. Calling this function group allows a user to click with the left mouse button and drag it to draw a rectangle.

The feature allows you to specify a dummy button ID whose associated section is the code to execute when the user releases the button press left mouse button. The last 4 parameters indicate the names of the variables that contain respectively the x and y coordinates of the upper left and lower right corner of the rectangle noted.

In addition to returning the coordinates of the rectangle, the last 4 parameters are used as input values for the function to specify the coordinates of the upper left corner and lower right corner of the rectangle that delimits the region of dialog on the user you can drag the mouse to draw.

The interpretation of the rest of the parameters are as follows:

Father%, referring to the dialog box. (The value returned by the call to w32GetParent).
button%, a value between 200 and 299 that designates the ID button number, real or fictional, whose associated section in $ SUB BtnClick subroutine code is executed when you release the left button press.
control%, name of the variable whose value will determine whether to allow the user to click and drag to draw. A value of 0 disables and 1 enables. Every time you send a notification to the designated button that the user has released the left mouse button, control% this variable is assigned the value 0.

The usual place where to place the call to this function is the end of the section $ SUB Init.

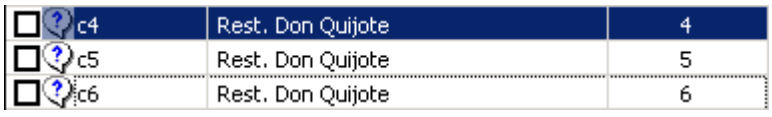
In all dialogues that make up an executable should not be called more than once to the same function in this group. Thus, to allow the user to define regions in 3 different dialogues that belong to the same project

Call w32DefineRegion1 ...
Call w32DefineRegion2 ...
Call w32DefineRegion3 ...

Note: Depending on the direction in which finally define the rectangle may return values for the coordinates of the left and right corners are exchanged each other.

w32ListSetFocusItem call (lis% item% addrof (ret%))

Sets the focus element of the list item% lis%, so that the user can operate the keys that produce a shift in the displayed items and the list selected. When an element has focus is surrounded by a dotted rectangle standard Focus. As shown in the figure, the focus is set to a list item unselected.



call w32ListSetHotItem (lis%,item%,addrof (ret%))

Enhances one element of the list. Each list can have only one element highlighted. This property holds even if all elements are deleted from the list and loaded again. To disable this property from the list must be set as the second parameter of the function the value -1.

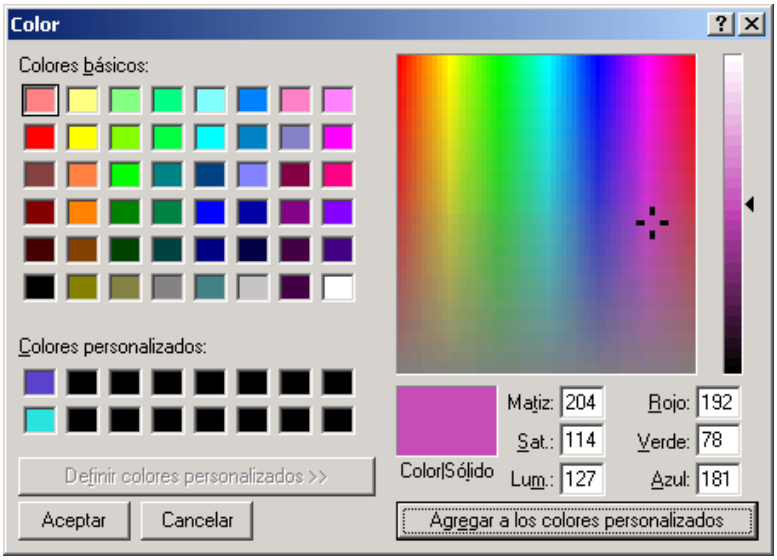
c6	Rest. Don Quijote	6
c7	Rest. Don Quijote	7

call w32CreateProgressH (padre%,x%,y%,ancho%,alto%,continua%,vertical%,addrof (hand%))

This function is identical to w32CreateProgress except that you add the initial parameter to specify the window or dialog on which to create the progress bar. This procedure avoids the need to put the focus on a control, and thus improves performance.

w32ChooseColor call (father%, r%, g%, b%, addrof (network%), addrof (green%), addrof (blue%), addrof (ret%))

Displays a dialog box so the user can select a color or define your own custom colors, which are stored in successive calls to this function is performed within the same instance of the program



The interpretation of the parameters is as follows:

Father%, is the reference to the main window active, must be the value returned by w32GetParent.

The following three parameters determine the color that will show up as selected when this dialogue.

r% - value of the red component of the color; Valid values are 0 .. 255
g% - value of the green component of the color; Valid values are 0 .. 255
b% - value of the blue component of the color; Valid values are 0 .. 255

The following three parameters determine the color that the user has chosen.

network% - value of the red component of the color; Valid values are 0 .. 255
green% - value of the green component of the color; Valid values are 0 .. 255
blue% - value of the blue component of the color; Valid values are 0 .. 255

Ret%, if the user pressed OK this parameter is set to 1.

w32DeleteObject call (Object%)

This function has the effect of freeing memory objects (mostly images and fonts) that are no longer needed in the execution of the program. The system running Windows applications released from memory all controls and objects belonging to a dialog when it is closed.

However, sometimes it is necessary to load resources in a dialogue that is no longer needed before this is closed, in these cases should be used to free w32DeleteObject function objects (mostly images and fonts) and function to release the w32DestroyWindow memory controls.

w32GetBitmapDimensions call ($ path, addrof (w%), addrof (high%))

Returns the width and height in pixels of BMP image whose path is specified as the first parameter. If there is no image width% values and high% will be 0.

w32CreateImageHEx call (hand%, file $, x%, y%, width%, height%, addrof (Picture%), addrof (control%))

This function has the same behavior as w32CreateImageH, except that it adds a penultimate parameter that returns a reference to the image so that if necessary can be released with a call to w32DeleteObject.

w32B3DH call (button%)

This function is identical except w32B3D is added to the initial parameter to specify the reference button (w32GetWindow value obtained). This procedure avoids the need to put the focus on a control, and thus improves performance.

w32BFlatH call (button%)

This function is identical except w32BFlat is added to the initial parameter to specify the reference button (w32GetWindow value obtained). This procedure avoids the need to put the focus on a control, and thus improves performance.

w32BMultilineH call (button%)

This function is identical except w32BMultiline is added to the initial parameter to specify the reference button (w32GetWindow value obtained). This procedure avoids the need to put the focus on a control, and thus improves performance.

w32BLoadImageH call (button%, $ file, width%, height%, addrof (Picture%))

This function is identical except w32BLoadImage is added to the initial parameter to specify the reference button (w32GetWindow value obtained). This procedure avoids the need to put the focus on a control, and thus improves performance. The return value is a reference to the image that can be deleted to free memory via w32DeleteObject function.

w32BFLoadImageH call (button%, $ file, width%, height%, addrof (Picture%))

This function is identical except w32BFLoadImage is added to the initial parameter to specify the reference button (w32GetWindow value obtained). This procedure avoids the need to put the focus on a control, and thus improves performance. The return value is a reference to the image that can be deleted to free memory via w32DeleteObject function.

w32ComboGetCurSel call (combo% addrof (sel%))

Returns the index of the item that is selected in the combo box whose reference is passed as the first argument. In case no item is selected returns a value of -1.

w32ComboResetContent call (combo% addrof (ret%))

Clears the contents of the combo box whose reference is passed as the first argument.

w32CreateCalendarH call (father%, x%, y%, width%, height%, addrof (cal%))

This function is identical to w32CreateCalendar except that you add the initial parameter to specify the window or dialog on which to create the calendar. This procedure avoids the need to put the focus on a control, and thus improves performance.

call w32CreateProcess ($ command, addrof (return%))

Run a WINDOWS or DOS command ensuring it will not run the next statement in the program until the command is completed. For calls to DOS commands or programs they should proceed by command.com / c, no DOS window appears in the taskbar.
% The return value indicates whether the command was executed successfully.

Example:

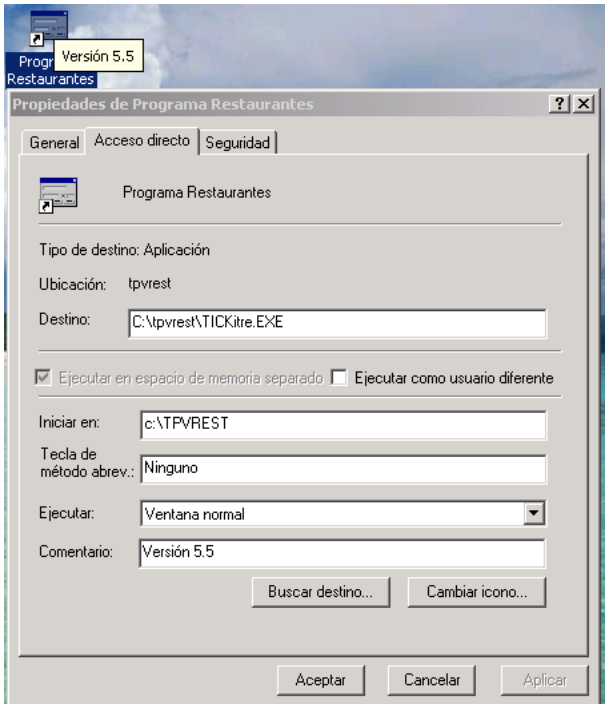
```
call w32CreateProcess ("command.com / c 900 3 sound" addrof (ret%))
```

w32CreateShortCut call ($ program, $ access, dir_trabajo $, $ comment, addrof (return%))

Create a shortcut to a program and puts $ this access to the file. Lnk whose path is specified as the second parameter. The third parameter specifies the working directory from where the program begins execution. Requires CETutils.dll file dated 19/03/2002 or later is located in the directory of the program that calls this function or any other directory pointed to by the PATH.

Ejemplo:

```
Rem Determinar la ruta del escritorio...Véase funciones del registro
call cRegOpenKey (80000001H,"Software",addrof (sub%),0,addrof (ret%))
call cRegOpenKey (sub%,"Microsoft",addrof (sub1%),0,addrof (ret%))
call cRegOpenKey (sub1%,"Windows",addrof (sub2%),0,addrof (ret%))
call cRegOpenKey (sub2%,"CurrentVersion",addrof (sub3%),0,addrof (ret%))
call cRegOpenKey (sub3%,"Explorer",addrof (sub4%),0,addrof (ret%))
call cRegOpenKey (sub4%,"Shell Folders",addrof (sub5%),0,addrof (ret%))
call cRegQueryValue (sub5%,"Desktop",addrof (donde$),addrof (ret%))
Rem – Fin de la determinación de la ruta del escritorio

call w32CreateShortCut ("c:\TPVREST\tickitre.exe",donde$&"\Programa
Restaurantes.lnk","c:\TPVREST","Versión 5.5",addrof (ret%))
```



call w32CreateTipH (control%,texto$,addrof (tip%))

This is identical to w32CreateTip with the proviso that the initial parameter is added to the control reference specify% (value obtained w32GetWindow). This procedure avoids the need to put the focus on a control, and thus improves performance.

call w32CreateWindowH (father% ID%, $ type, style% estiloex%, title $, x%, y%, width%, height%, addrof (control%))

Capítulo 11 CALLS FROM WIN32 API CET W32

This feature allows you to create controls at runtime WINDOWS various types.

The meaning of the parameters is as follows:
Father%, referring to the dialog must be the value returned by w32GetParent.
ID%, number to be used to identify the control to be created. It's the same identify using standard controls W32 CET.
$ Type is a string that identifies the type of control to create. Some examples: "Edit", "Button", "SysDateTimePick32". (See appendix to this function)
Style%, determine the style that will be in control. Usually contains 1 or more constants specified below to which is applied the OR operator.

EstiloEx%, determines the extended style that will be in control. Usually contains 1 or more constants specified below to which is applied the OR operator.

$ Title, is the initial value of text to display the control (if applicable).
Control% reference to the control that has been created.

Ejemplo:

```
call w32CreateWindowH (padre%,106,"SysDateTimePick32",WS_BASE% OR WS_TABSTOP% OR WS_GROUP%
OR DTS_SHOWNONE%, WS_EX_CLIENTEDGE%, "", 470, 60, 300, 30, addrof (r106%))

call cSetFocus (dlg1HDLG%,106)
```

It has created a control to select the date (allowing the user to specify a null date) and then put the focus on el.

Annex:

```
Generic Window Styles Rem
WS_OVERLAPPED% = 00000000H
Rem Create an overlay window, which normally has a title and a border.
WS_POPUP% = 80000000H
Rem Create a popup. Can not be used with WS_CHILD%
WS_CHILD% = 40000000H
Rem Creates a child window (control). Can not be used with WS_POPUP%
WS_VISIBLE% = 10000000H
Rem Creates a window initially visible.
WS_DISABLED% = 08000000H
Rem Creates a window initially disabled
WS_MINIMIZE% = 20000000H
Rem Creates a window that is initially disabled. To use just by the style WS_OVERLAPPED%
WS_MAXIMIZE% = 01000000H
Rem Creates a window of maximum size
WS_CAPTION% = 00C00000H
Rem Creates a window that has a title bar (implies the style WS_BORDER%). Can not be used
with WS_DLGFRAME%
WS_BORDER% = 00800000H
Rem Creates a window that has a border.
WS_DLGFRAME% = 00400000H
Rem Creates a window with double border but no title.
WS_VSCROLL% = 00200000H
Rem Creates a window that has a vertical scroll bar
WS_HSCROLL% = 00100000H
Rem Create a window with a horizontal scroll bar
```

```
WS_SYSMENU% = 00080000H
Rem Creates a window that has a control box in your menu bar título.Se used only with
windows title bar
WS_MINIMIZEBOX% = 00020000H
Rem Creates a window that has a minimize button
WS_MAXIMIZEBOX% = 00010000H
Rem Creates a window that has a maximize button
WS_GROUP% = 00020000H
Rem Specifies the first control of a group of controls in which the user can move from one
to another with the cursor keys. It is understood that all controls created after the first
control that do not have defined this style in the same group. The next control is created
with the style WS_GROUP% starts the next group (ie a group ends when the next begins)
WS_TABSTOP% = 00010000H
Rem The user can access this control using the tab key
WS_BASE% = (WS_CHILD WS_VISIBLE OR%%)
Styles Rem basic building controls


Extended Window Styles Rem
WS_EX_CLIENTEDGE% = 00000200H
Rem Specifies that the window has a border with the enhanced edge.


Rem static controls. Type: "Static"
SS_LEFT% = 00000000H
SS_CENTER% = 00000001H
SS_RIGHT% = 00000002H
SS_ICON% = 00000003H
SS_BLACKRECT% = 00000004H
SS_GRAYRECT% = 00000005H
SS_WHITERECT% = 00000006H
SS_BLACKFRAME% = 00000007H
SS_GRAYFRAME% = 00000008H
SS_WHITEFRAME% = 00000009H
SS_SIMPLE% = 0000000BH
SS_LEFTNOWORDWRAP% = 0000000CH
SS_NOPREFIX% = 00000080H
SS_NOTIFY% = 00000100H

Buttons Rem. Type: "Button"
BS_PUSHBUTTON% = 00000000H
BS_DEFPUSHBUTTON% = 00000001H
BS_CHECKBOX% = 00000002H
BS_AUTOCHECKBOX% = 00000003H
BS_RADIOBUTTON% = 00000004H
BS_3STATE% = 00000005H
BS_AUTO3STATE% = 00000006H
BS_GROUPBOX% = 00000007H
BS_USERBUTTON% = 00000008H
BS_AUTORADIOBUTTON% = 00000009H
BS_OWNERDRAW% = 0000000BH
BS_LEFTTEXT% = 00000020H
BS_MULTILINE% = 00002000H
BS_VCENTER% = 00000C00H

Rem edit controls. Type: "Edit"
ES_LEFT% = 00000000H
ES_CENTER% = 00000001H
ES_RIGHT% = 00000002H
ES_MULTILINE% = 00000004H
ES_UPPERCASE% = 00000008H
ES_LOWERCASE% = 00000010H
ES_PASSWORD% = 00000020H
ES_AUTOVSCROLL% = 00000040H
ES_AUTOHSCROLL% = 00000080H
ES_NOHIDESEL% = 00000100H
ES_OEMCONVERT% = 00000400H
ES_READONLY% = 00000800H
ES_WANTRETURN% = 00001000H
ES_NUMBER% = 00002000H

Scroll Bar Rem. Type: "ScrollBar"
SBS_HORZ% = 0000H
SBS_VERT% = 0001H
SBS_TOPALIGN% = 0002H
SBS_LEFTALIGN% = 0002H
SBS_BOTTOMALIGN% = 0004H
SBS_RIGHTALIGN% = 0004H
SBS_SIZEBOXTOPLEFTALIGN% = 0002H
SBS_SIZEBOXBOTTOMRIGHTALIGN% = 0004H
SBS_SIZEBOX% = 0008H

Rem List Box. Type: "ListBox"
```

```
LBS_NOTIFY% = 0001H
LBS_SORT% = 0002H
LBS_NOREDRAW% = 0004H
LBS_MULTIPLESEL% = 0008H
LBS_OWNERDRAWFIXED% = 0010H
LBS_OWNERDRAWVARIABLE% = 0020H
LBS_HASSTRINGS% = 0040h
LBS_USETABSTOPS% = 0080H
LBS_NOINTEGRALHEIGHT% = 0100H
LBS_MULTICOLUMN% = 0200H
LBS_WANTKEYBOARDINPUT% = 0400H
LBS_EXTENDEDSEL% = 0800H
LBS_DISABLENOSCROLL% = 1000H
LBS_NOSEL% = 4000H
LBS_STANDARD% = (OR WS_VSCROLL LBS_NOTIFY%%% WS_BORDER OR OR OR WS_HSCROLL%
LBS_USETABSTOPS%)

Rem Combo Box. Type: "ComboBox"
CBS_SIMPLE% = 0001H
CBS_DROPDOWN% = 0002H
CBS_DROPDOWNLIST% = 0003H
CBS_OWNERDRAWFIXED% = 0010H
CBS_OWNERDRAWVARIABLE% = 0020H
CBS_AUTOHSCROLL% = 0040h
CBS_OEMCONVERT% = 0080H
CBS_SORT% = 0100H
CBS_HASSTRINGS% = 0200H
CBS_NOINTEGRALHEIGHT% = 0400H
CBS_DISABLENOSCROLL% = 0800H



Styles Rem Date Time Picker control. Type: "SysDateTimePick32"
DTS_TIMEFORMAT% = 0009H
Rem control displays time date time
DTS_SHOWNONE% = 0002H
Rem It indicates that the user is allowed to specify no value for the date / time by
uncheck the check box that is displayed in the same.
```

call w32DTPSetSel (control%,nulo%,ano%,mes%,dia%,hora%,min%,seg%,addrof (ret%))

Sets the value assigned to a control date and time. The interpretation of the parameters is as follows:

Control% control identifier Date / Time. (Return value w32CreateWindowH)
Null% indicator that establishes whether to specify a null value. If this parameter is assigned a value of 1 disables the control content, if you are assigned a 0 control content is active according to the values `of the following parameters.`

The year value must be a 4 digit number.

call w32DTPGetSel (control% addrof (year%), addrof (month%), addrof (day%), addrof (time%), addrof (min%), addrof (s%), addrof (ret%))

Gets the value specified for the control date / time.
The return value is 0 if the content is not disabled.

call w32EditGetLineCount (control% addrof (few%))

Returns the number of lines of text that make up the value of the multiline edit control specified by parameter control%.

call w32EditGetLine (control% line% addrof ($ text), addrof (length%))

Returns the control text% owned multi-line% specified as a parameter (starting from 0). The length% return parameter contains the number of valid characters in the text string $.

w32GetDiskFreeSpaceEx ($ unit, addrof (bytesfree $), addrof (retcode%).

Returns the number of bytes that are free on the drive that you have specified. This function has been introduced for large disk operations.

call w32GetDiskSpaceEx ($ unit, addrof ($ space), addrof (ret%))
Returns the total drive capacity in bytes. This function has been introduced for large disk operations.

call w32GetVersion (addrof (see%))
Determines at run time if you are working on a platform 95/98/Me or NT/2000/XP.

Return:
1 - 95/98/Me
2 - NT/2000/XP

call w32GetWindowID (control% addrof (id%))
Returns the identifier for the control% using standard controls W32 CET.

call w32GetWindowRectEx (control% addrof (L%), addrof (top%), addrof (der%), addrof (background%), addrof (ret%))
Returns the coordinates, relative to the dialog box, the position of the upper and lower control% is specified as a parameter.

call w32ImageBackgroundn (father%, $ image, width%, height%, addrof (Skin%))
Where n is a value from 1 to 9. Calling this function group to define the background wallpaper that displays a dialog box.

The interpretation of the parameters is as follows:

Father%, reference to the dialog. (Value returned by the function w32GetParent)
$ image, file path. bmp to be displayed as wallpaper.
% width - Width in pixels of the image bmp. In this first version you specify 8.
high% - High in bmp image pixels. In this first version you specify 8.
Skin% - Is the ID of the image resulting tapestry. This value can be used to make static controls display transparently. (See w32SetCtrlColorn function)

The usual place where to place the call to this function is the end of the section $ SUB Init.

In all dialogues that make up an executable should not be called more than once to the same function in this group. Thus, to define the three dialogues Skin functions to be used ...

Call w32ImageBackground1 ...
Call w32ImageBackground2 ...
Call w32ImageBackground3 ...

w32LBoxResetContentH call (control%)
This is identical to w32LBoxResetContent with the proviso that the initial parameter is added to the control reference specify% (value obtained w32GetWindow). This procedure avoids the need to put the focus on a control, and thus improves performance.

w32ListGetRect call (lis% item% subitem% addrof (L%), addrof (top%), addrof (der%), addrof (background%), addrof (ret%))
Returns the coordinates relating to the current position of the list lis% of the upper and lower values determined by field item% and subitem%.

w32ListSetColumn call (list%% column, $ title, addrof (ret%))
Change the title of a column in a list.
The column value% is the column index (starting from 0) $ whose title you want to change.

w32ListSetFullBkColor call (lis% r% v%, y%)

Sets the background color of the list items lis% with no text.
r% - value of R (red) color; Valid values are 0 .. 255
v% - component value V (green) color; Valid values are 0 .. 255
a% - value component (blue) color; Valid values are 0 .. 255

w32MAPISendMail call ($ subject, $ content, $ direccion_destino, nombre_destino $, numero_ficheros%, $ file, addrof (ret%))

Enables sending email are user intervention.
The syntax is:

$ Subject, Subject of e-mail
$ Content, email content
$ Direccion_detino, e-mail address, beginning with SMTP (eg SMTP: job@omegasystems.com)
$ Nombre_destino, Alternative name to specify the person for whom the e-mail
numero_ficheros%, number of files attached. Can be 0 or 1.
$ file, full path of the file to attach. If the parameter is 0 numero_ficheros%, discard any submission attached.
ret% return value. A value other than 0 indicates error.

Example:

```
w32MAPISendMail call ("e-mail auto", "Look how well I write", "SMTP: job@omegasystems.com",
"Job Rodriguez", 1, "c: \ winnt \ system32 \ autoexec.nt" addrof ( ret%))
```

w32CreateMenu call (addrof (menu%))

Create a reference to a context menu. The return value should be used for any operation you want to perform with the menu.

call w32MenuInsertItem (menu%, id%, tipo%, texto$, addrof (ret%))

Add a menu item to the end of%.

The interpretation of the parameters is as follows:
Menu% is the identifier of the menu to which you add the item. (Must be the value returned by w32CreateMenu)

ID%, ID that is associated with the menu item. It is intended to help determine the selection made by the user. In case the input is in turn a submenu, the value of this parameter must be the same identifier.

Type%, is a value between 0 and 2 which determines the type of input that is being inserted.
0, The item is a selectable item inserted normal.
1, The item to be inserted is itself another menu (submenu)
2, The item to be inserted is a separation line unselectable.

$ Text, text string to be displayed in the entrance.

w32MenuItemBitmap call (menu% ID%, $ file, addrof (Picture%))

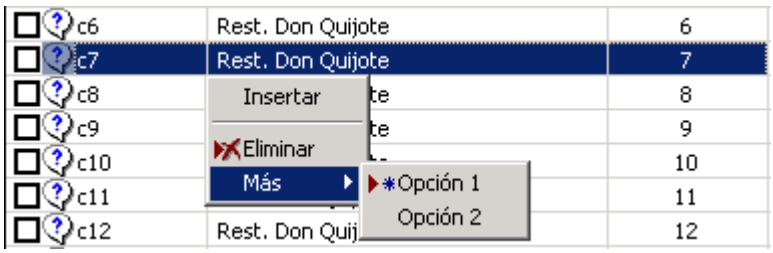
Associates a file $. Bmp to menu item whose identifier% (% ID) is specified as a parameter. The size of the image. Bmp is 13 x 13 pixels to be displayed to the left of the text pertaining to the entry. The return value of the function is a reference to the image that can be deleted to free up memory w32DeleteObject.

w32MenuPopup call (menu% control%, LH%, top%, addrof (sel%))

This is the only function regarding menus that have a visual effect. % Presents the menu as a parameter specified on the control% indicated. The above parameters and left%% determine the coordinates, relative to the dialog box, in the left corner where the menu is displayed. The return value is the identifier (ID%) of the menu item that the user selected.
w32MenuDestroy call (menu% addrof (ret%))
Destroy all references to menu% (and all its submenus) is specified as a parameter. This feature is intended to free up memory.

Ejemplo de funciones de menús contextuales:

```
$SUB dlg1Init
REM     Initialize dialog save area
REM@# inistart
.
.
.
call w32CreateMenu (addrof (submenu%))
call w32MenuInsertItem (submenu%,6,0,"Opción 1",addrof (ret%))
call w32MenuInsertItem (submenu%,7,0,"Opción 2",addrof (ret%))

call w32CreateMenu (addrof (menu%))
call w32MenuInsertItem (menu%,1,0,"Insertar",addrof (ret%))
call w32MenuInsertItem (menu%,2,2," ",addrof (ret%))
call w32MenuInsertItem (menu%,3,0,"Eliminar",addrof (ret%))
call w32MenuInsertItem (menu%,submenu%,1,"Más",addrof (ret%))

Rem ancho 13, alto 13
call w32MenuItemBitmap (menu%,3,".\eliminar.bmp",addrof (ret%))
call w32MenuItemBitmap (menu%,6,".\insertar.bmp",addrof (ret%))
.
.
.
call w32ListGetEvents2 (padre%,lis%,250,addrof (evento%),addrof (l_item%),addrof (l_sub%))

REM@# iniend
$EXIT

$SUB dlg1BtnClick
REM     Button Click save area
REM@# btnstart
SELECT CMDID%

CASE 250
        Rem Notificaciones para la lista lis%
        Rem evento%->Numero de evento
        Rem l_item%->item de la lista  (fila)
        Rem l_sub%->Sub-item de la lista  (columna)
SELECT evento%

CASE 4
Rem Click con el botón de derecho en un campo
call w32GetWindowRectEx (lis%, addrof (izq%), addrof (alto%), addrof (der%), addrof
(fondo%),addrof (ret%))
desx%=izq%
desy%=alto%
Rem Coordenadas relativas a la lista, por eso se aplica desplazamiento
call w32ListGetRect (lis%, l_item%, l_sub%, addrof (izq%), addrof (alto%), addrof (der%),
addrof (fondo%),addrof (ret%))
donde%=fondo%+desy%
call w32MenuPopup (menu%,lis%,izq%+desx%,donde%,addrof (sel%))
        SELECT sel%
        CASE 3
                Call w32ListDeleteItem (lis%,l_item%,addrof (ret%))
        .
        .
        .
        CEND


CEND
CEND
REM@# btnend
GOSUB dlg1.BTNEND
$EXIT
```



call w32SetWindowOrder (control%,control_ref%,addrof (ret%))

Sets the control% specified as the first argument will be next on the control tab order is specified as the second argument. Should this second parameter is 0, the control will the first in the tab order of the dialog**.**

call w32TabStopWindow (control% si_no%)

Determines whether the control specified as the first argument% will be able to be accessed by the user by pressing the Tab key. If the second argument is 0, the control is no longer part of the list of controls that is crossed with the Tab key.

call w32TipUpdateTextH (tip%, control% $ nuevotexto, addrof (return%))

This function is identical to except w32TipUpdateText inserting a second parameter to specify the control reference% (value obtained with w32GetWindow). This procedure avoids the need to put the focus on a control, and thus improves performance.

w32TreeGetRect call (tree% position% addrof (L%), addrof (top%), addrof (der%), addrof (background%), addrof (ret%))

Returns the coordinates relating to the current position of the tree tree%, of the upper and lower corner of the element determined by the reference position%.

w32WinHelp call (father% $ fichero_ayuda, operation%, $ value, addrof (ret%))

Enables presents options for a help file. Hlp.

The interpretation of the parameters is as follows:

Father%, is the identifier of the dialog box from which you launch the function call
Fichero_ayuda $, filename. Hlp
Operation%, is the code of the display mode to be used (see examples).
$ Value is an additional parameter on the operation%. If you do not have to specify it will "".
Ret%, return value. 0 indicates error.

Examples:

```
Rem 11 presents the search index
w32WinHelp call (father%, "videos31.hlp", 11, "", addrof (ret%))
Rem 4 Help on using the help
w32WinHelp call (father%, "videos31.hlp", 4, "", addrof (ret%))
Rem 257 Search a keyword
w32WinHelp call (father%, "videos31.hlp", 257, "CINTES VIDEO" addrof (ret%))
```

w32ListGetColumnsOrder call (lis%, cols% addrof ($ string), addrof (ret%))

Returns the current order of the columns in the list lis% according to the drag-and-drop operations that the user may have done. Cols% parameter should be the number of columns in the list, and the return value string $ contain comma-separated list of number of columns in the order specified by the user. Requires CETutils.dll file dated 25/03/2002 or later.

w32CreateGrid call (father%, $ title, ID%, x%, y%, width%, height%, addrof (grid%))

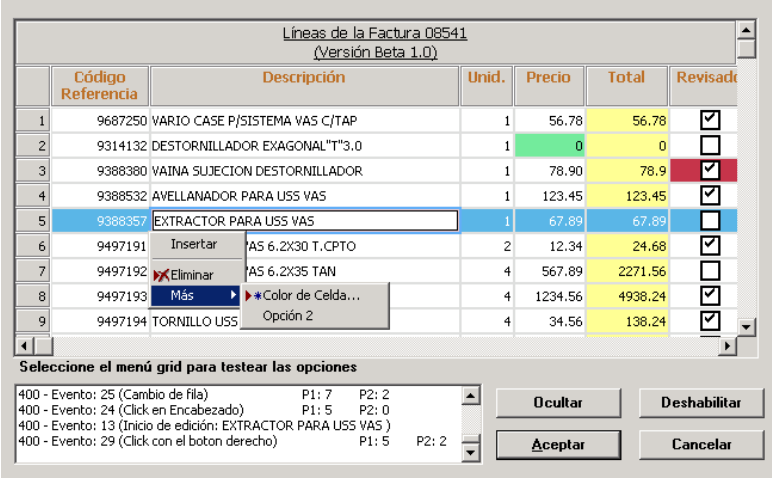
Create a Grid.
The meaning of the parameters is as follows:

Father%, referring to the dialog must be the value returned by w32GetParent.
$ Title, Text to appear in the title row of the control
ID%, number to be used to identify the control and capture events. It must be different for each of the grids that are included in the same executable.
Grid%, reference to the control grid created.

Limitations:
- Can be created simultaneously to a maximum of 20 control grid in the same executable.
- The maximum number of rows that can have a grid is 32000
- The maximum number of columns that can have a grid is 256

- The maximum number of characters that can be entered in a cell is 150

Requires CETgrid.dll file or later date 23/04/2003.


Líneas de la Factura 08541
(Versión Beta 1.0)
Código Referencia
Descripción
Unid.
Precio
Total
Revisad
9687250 VARIO CASE P/SISTEMA VAS C/TAP 1 56.78 56.78
9314132 DESTORNILLADOR EXAGONAL"T"3.0 1 0 0
9388380 VAINA SUJECION DESTORNILLADOR 1 78.90 78.9
9388532 AVELLANADOR PARA USS VAS 1 123.45 123.45
9388357 EXTRACTOR PARA USS VAS 1 67.89 67.89
9497191 'AS 6.2X30 T.CPTO 2 12.34 24.68
9497192 'AS 6.2X35 TAN 4 567.89 2271.56
9497193 4 1234.56 4938.24
9497194 TORNILLO USS 4 34.56 138.24
Insertar
Eliminar
Más
Color de Celda...
Opción 2
Seleccione el menú grid para testear las opciones
400 - Evento: 25 (Cambio de fila) P1: 7 P2: 2
400 - Evento: 24 (Click en Encabezado) P1: 5 P2: 0
400 - Evento: 13 (Inicio de edición: EXTRACTOR PARA USS VAS)
400 - Evento: 29 (Click con el boton derecho) P1: 5 P2: 2
Ocultar
Deshabilitar
Aceptar
Cancelar

Ejemplo: Grid.prj

call w32GridDeleteAllItems (grid%)

Deletes all rows in the grid control referenced by the first parameter.
Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridDeleteColumn call (grid%, col%)

Deletes the column grid (starting from 1) is specified as the second parameter. Can not delete column 1.
Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridDeleteItem call (grid% temp%)

Delete the grid row (starting from 1) is specified as the second parameter.
Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridEditCurCell call (grid%)

Starts editing the currently selected grid cell. The user can start editing any cell by pressing F2 or double clicking on the character of the cell where you want to position the cursor.

Quick Edit:
a) Another possibility to modify the contents of a cell is passed to type a value.
b) If the cell selected then press the space bar adds a space and place the cursor at the end of the text.
c) If the selected cell on pressing the DEL key (Backspace) erases the contents hereof and allows editing.

You have changed the contents of a cell by any of the three methods so quick edit keys Home, End and Delete keys have no effect. Also the cursor left and right cursor will cause the adjacent cell is selected.

In any type of editing you can press the Escape key to undo the data entry.
Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridEditing call (grid% addrof (editing%))

Returns 1 if a cell is being edited in the grid.
Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridGetCellProtection call (grid%, row%, col% addrof (protected%))

Returns 1 if the grid cell parámentros referenced by row and col%% is protected. Protected cells are those that the user can not edit, and whose background color can be highlighted.
Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridGetCol call (grid% addrof (col_actual%))

Returns the column number of the cell in the grid that is currently selected.
Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridGetColCount call (grid% addrof (num_col%))

Returns the number of columns in the grid.
Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridGetEditString call (grid% addrof ($ txt))

This feature should be invoked when previously been found by w32GridEditing, which is a cell in editing, in which case the return parameter will contain the current value thereof.
Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridGetEvent call (grid% addrof (event%))

Returns the number of events that generated the grid, and w32GridEventP2 w32GridGetEventP1 functions should call ahead to get the value of the first and second property of the event that just occurred. Below is a table of values:

Evento	Descripción	Propiedad 1	Propiedad 2
1	Pulsación de *F1*	Fila actual	Columna actual
2	Pulsación de *F2* (No se notifica, está reservado para editar)		
3	Pulsación de *F3*	Fila actual	Columna actual
4	Pulsación de *F4*	Fila actual	Columna actual
5	Pulsación de *F5*	Fila actual	Columna actual
6	Pulsación de *F6*	Fila actual	Columna actual
7	Pulsación de *F7*	Fila actual	Columna actual
8	Pulsación de *F8*	Fila actual	Columna actual
9	Pulsación de *F9*	Fila actual	Columna actual
10	Pulsación de *F10* (No se notifica, está reservada por WINDOWS para activar menús)		
11	Pulsación de *F11*	Fila actual	Columna actual
12	Pulsación de *F12*	Fila actual	Columna actual
13	Se ha iniciado la edición de una celda.	Fila	Columna
14	Ha concluido la edición de una celda	Fila	Columna
15	Pulsación de *Suprimir*	Fila actual	Columna actual
16	Pulsación de *Insertar*	Fila actual	Columna actual
17	Pulsación de *Inicio*	Fila actual	Columna actual
18	Pulsación de *Fin*	Fila actual	Columna actual
19	Pulsación de *Cursor Arriba*. (Sólo se notifica cuando se está en la primera fila)	1	Columna actual
20	Pulsación de *Cursor Abajo*. (Sólo se notifica cuando se está en la última fila)	Última Fila	Columna actual
21	Pulsación de *Re Pág*. (Sólo se notifica cuando se está en la primera fila)	1	Columna actual
22	Pulsación de *Av Pág*. (Sólo se notifica cuando se está en la	Última Fila	Columna actual

	última fila)		
23	No se utiliza. (Reservado)		
24	Clic fuera del área de celdas.	Fila Correspondiente, -1 en otro caso	Columna correspondiente, -1 en otro caso
25	Se ha seleccionado una celda perteneciente a una fila distinta a la anterior	Fila anterior	Columna anterior
26	Se ha seleccionado una celda perteneciente a una columna distinta a la anterior	Fila anterior	Columna anterior
27	Se ha puesto el foco sobre el control	ID del control que anteriormente tenía el foco ó 0 si el control pertenecía a otro diálogo	-1
28	El control ha perdido el foco	ID del control que pasa a recibir el foco ó 0 si el control no pertenece al diálogo	-1
29	Clic con el botón derecho.	Fila Correspondiente, -1 en otro caso	Columna correspondiente, -1 en otro caso
30	Doble-Clic fuera del área de celdas.	Fila Correspondiente, -1 en otro caso	Columna correspondiente, -1 en otro caso
31	Pulsación de ENTER (Sólo se notifica cuando no hay ningún desplazamiento asociado a esa tecla. Ver *w32GridSetEnterMode*)	Fila actual	Columna actual
32	Pulsación de ESCAPE (Sólo se notifica si no se está editando ninguna celda)	Fila actual	Columna

From 9.32m version, the runtime system can process CETW32 special event number 15 by which they can detect the notifications generated by the compiler controls added, as is the case of the Grid. The value of the variable reserved CMDID% contain the ID of the control that generated the event.

So the call to this function w32GridGetEvent well as associated functions and w32GridEventP2 w32GridGetEventP1 must occur within the subroutine Event filtering and event 15 and CMDID% value corresponding to the ID that is associated with the grid in the creation by w32CreateGrid.

Requires CETgrid.dll file or later date 23/04/2003.

Ejemplo:

```
$SUB grid1Event
SELECT CET_EVTYPE%
REM     Event notification save area
REM@# evtstart
CASE 15
        Rem Eventos Especiales....
        SELECT CMDID%
        Rem 400 es el ID del control Grid% asignado en w32CreateGrid
        CASE 400
                call w32GridGetEvent (grid%,addrof (ev%))
                SELECT ev%
                CASE 9
                        call w32GridGetEventP1 (grid%,addrof (p1%))
                        call w32GridGetEventP2 (grid%,addrof (p2%))
```

```
                    call w32GridGetText (grid%,p1%,p2%,addrof (txt$))
                    texto$="  (F9 pulsado: "&txt$&" ) "

            CEND


       CEND
REM@# evtend
CEND
$EXIT


REM
REM     The dialog is being initialized
REM     Set the initial contents and state of the list box (s)
REM     and other controls
REM

$SUB grid1Init
REM     Initialize dialog save area
REM@# inistart
di%=grid1HDLG%
call cSetFocus (di%,200)
call w32GetParent (addrof (padre%))
call w32CreateGrid (padre%,"Líneas de la Factura 08541"&chr$ (13)&" (Versión Beta
1.0)",400,10,15,625,285,addrof (grid%))
```

call w32GridGetEventP1 (grid%, addrof (p1%))

Returns the first property associated with the last event occurred. (See w32GridGetEvent)

Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridGetEventP2 call (grid% addrof (p2%))

Returns the second property associated with the last event occurred. (See w32GridGetEvent)

Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridGetItemCount call (grid% addrof (row%))

Returns the number of rows in the control grid is specified as the first parameter.

Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridGetRect call (grid% fil%, col% addrof (L%), addrof (high%), addrof (der%), addrof (background%))

Returns coordenas in pixels, relative to the position of the grid, the rectangle that bounds the cell defined by the row and column as the second and third parameters.

The return values define respectively the horizontal and vertical coordinates of the upper left corner and lower right corner of the rectangle.

Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridGetRow call (grid% addrof (row%))

Returns the row number belonging to the currently selected cell in the Grid.

Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridGetText call (grid% fil%, col% addrof ($ text))

Returns the contents of the cell in the grid defined by the row and column as the second and third parameters.

Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridInsertColumn call (grid% position%)

Inserts an empty column at the specified position as the second parameter.

Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridInsertItem call (grid% position%)
Inserts an empty row at the specified position as the second parameter.

Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridIsHilighted call (grid% addrof (si_no%))
Returns a zero if it is enabled to shade the entire row when you select a cell belonging to it. (See also the documentation of the function w32ShowHilight).

Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridProtectCell call (grid% fil%, col%, protect%)
Protect or unprotect the cell of the grid defined by the row and column specified depending on whether the value of the last parameter is 0 or 1. When a cell is protected not allowing you to change the content and this is signaled by a special background color.

Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridSetAllowColResize call (grid%, resize%)
Allows or does not allow the user resizes the columns depending on whether the final parameter value is 0 or 1. By default if the user is allowed to change the column width.

Requires CETgrid.dll file dated 23/04/2003 or later.

call w32GridSetBkColor (grid% r% v%, y%)
Sets the background color of the cells in the grid%.
r% - value of R (red) color; Valid values are 0 .. 255
v% - component value V (green) color; Valid values are 0 .. 255
a% - value component (blue) color; Valid values are 0 .. 255

Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridSetCellBkColor call (grid% fil%, col%, r%, v%, y%)
Sets the background color of the cell in the control grid% defined by the row and column specified as the second and third parameters.

r% - value of R (red) color; Valid values are 0 .. 255
v% - component value V (green) color; Valid values are 0 .. 255
a% - value component (blue) color; Valid values are 0 .. 255

Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridSetColMaxChars call (grid%, column% max_char%)
The last parameter sets the maximum number of characters allowed the user to enter in the cells corresponding to the column of the grid that is specified as the second parameter.

In any case, never be allowed to enter more than 150 characters in a cell.

Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridSetColsNumbered call (grid% auto_numerar%)
Autoenumera or columns depending on whether the value of the last parameter of the grid is 1 or 0. The default behavior is that the columns are not autoenumeren, so that you can specify the title of each column by w32GridSetText function indicating the row number 0 corresponding to the row of column headers.

If you set the columns to be displayed autoenumeren then led them in Microsoft Excel.

Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridSetColWidth call (grid%, col%, width%)

Sets the width in pixels that the column will be specified as the second parameter. The number 0 is the column that shows the number or title of each row.

Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridSetCursorColor call (grid% r% v%, y%)

Sets the color to have the cursor or active cell delimiter.

r% - value of R (red) color; Valid values are 0 .. 255
v% - component value V (green) color; Valid values are 0 .. 255
a% - value component (blue) color; Valid values are 0 .. 255

Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridSetCursorPos call (grid%, row%, column%)

Set as active cell which is defined by the row and column specified as the second and third parameters.

Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridSetDim call (grid% num_filas% num_columnas%)

Sets the number of rows and columns that you will have the grid. If the number of rows or columns or current is less than ordered, will be added or deleted rows or columns respectively necessary so that the grid passes have dimensions specified as a parameter.

Requires CETgrid.dll file or later date 23/04/2003.

call w32GridSetEditable (grid%, editable%)

Allows the user to edit the cell contents of the grid depending on whether the value of the last parameter is 1 or 0. By default all cells are editable by the user except those defined as protected.

Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridSetEnterMode call (grid%, so%)

Defines whether to change the active cell when the user presses the ENTER key, according to the following possible values for the second parameter:

1 The selection will move to the cell to the right or to the next row if no more columns to the right. In any case never selects a cell belonging to a column whose width is 0. This mode of displacement is the one with the default grid.

2 selection will move to the cell in the same column but in the next row.

3 Not going to change the active cell or selected when the user presses ENTER. In this case the grid will generate a notification event of pressing the ENTER key. (Event No. 31)

Requires CETgrid.dll file dated 23/04/2003 or later.

call w32GridSetGridLineColor (grid% r% v%, y%)

Sets the color of the grid lines that delimit the cells.
r% - value of R (red) color; Valid values are 0 .. 255
v% - component value V (green) color; Valid values are 0 .. 255
a% - value component (blue) color; Valid values are 0 .. 255

Requires CETgrid.dll file dated 23/04/2003 or later.

call w32GridSetHeaderHeight (grid%, height%)

Sets the height of the row in pixles number 0, ie the row that displays column headers.

Requires CETgrid.dll file dated 23/04/2003 or later.

call w32GridSetHeaderTextColor (grid% r% v%, y%)

Sets the text color of the headings or column headings.
r% - value of R (red) color; Valid values are 0 .. 255
v% - component value V (green) color; Valid values are 0 .. 255
a% - value component (blue) color; Valid values are 0 .. 255

Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridSetHeadingFont call (grid%, $ source, points%, position% bold% italic% sub% strikeout%)

Set the font and its attributes for text in headers or column headers. The orientation parameter designates the degrees of inclination of the text. The last 4 parameters take value 1 to enable and 0 to disable attribute.

Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridSetProtectColor call (grid% r% v%, y%)

Sets the background color with which to emphasize that a cell is protected.
r% - value of R (red) color; Valid values are 0 .. 255
v% - component value V (green) color; Valid values are 0 .. 255
a% - value component (blue) color; Valid values are 0 .. 255

Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridSetRowHeight call (grid%, height%)

Set the height in pixels that will have all rows editable grid.

Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridSetRowsNumbered call (grid% auto_numerar%)

Autoenumera rows or not depending on whether the value of the last parameter of the grid is 1 or 0. The default behavior is that the rows are autoenumeren.

This function can be used so that it is possible to specify an alternative description for each row by the column indicating w32GridSetText function number 0 that corresponds to the column showing the number or title of each row.

Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridSetSelColor call (grid% r% v%, y%)

Sets the color to be the background of the entire row when you select a cell belonging to it, in case it has been determined that shade the entire row for the active cell. (See w32GridShowHilight)
r% - value of R (red) color; Valid values are 0 .. 255
v% - component value V (green) color; Valid values are 0 .. 255
a% - value component (blue) color; Valid values are 0 .. 255

Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridSetSelTextColor call (grid% r% v%, y%)

Sets the color to have the text of the entire row when selecting a cell belonging to it, in case it has been determined that shade the entire row for the active cell. (See w32GridShowHilight)
r% - value of R (red) color; Valid values are 0 .. 255
v% - component value V (green) color; Valid values are 0 .. 255
a% - value component (blue) color; Valid values are 0 .. 255

Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridSetText call (grid%, row%, column%, $ text)

Sets the cell contents of the grid defined by the row and column indicated as second and third parameter. The content specified as the last parameter must not be an empty string.

Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridSetTitleFont call (grid%, $ source, points%, position% bold% italic% sub% strikeout%)

Set the font and its attributes to the grid title text. The orientation parameter designates the degrees of inclination of the text. The last 4 parameters take value 1 to enable and 0 to disable attribute.

Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridSetTitleHeight call (grid%, height%)

Sets the height in pixels of the row reserved for displaying the title that has been assigned to the grid by w32CreateGrid.

Requires CETgrid.dll file dated 23/04/2003 or later.

w32GridShowHilight call (grid% resaltar_fila%)

Sets or not you highlight the entire row when you select a cell belonging to the same depending on the value assigned to the second parameter is 1 or 0.

By default NOT highlight the entire row. (W32GridIsHilighted function to check which mode is currently active)

Requires CETgrid.dll file dated 23/04/2003 or later.

w32EditENTEREx call (index% hand% ID%, addrof (key%))

The call to this function allows the user to validate the information entered in any edit box
a dialogue by pressing the ENTER key, so that the focus goes to
next control. This special behavior only affects the editing controls to
which they are found associated function.
Furthermore it can process any key pressed by the user.

This instruction implements the same functionality as w32EditENTER, provided that notification of events keystrokes done differently to prevent some unwanted effects of change in focus that occurred sometimes when you used the previous version . Requires 9.32ma version from which the runtime system to process the event CETW32 special number 15 by which it can detect additional notifications. The value of the variable reserved CMDID% contain the ID of the control that generated the event.

The interpretation of the parameters is as follows:

index% integer value between 1 and 99
Hand%, referring to the edit control. (Value obtained with w32GetWindow)
ID%, between 100 and 199 that designates the control handle.
key% variable name that will contain the value of the key pressed in every moment. (For a full account of the values of the keys see w32skeybd_event function documentation)

The usual place where to place the call to this function is the end of the section $ SUB Init.

In all dialogues that make up an executable to use a different value for each index% edit control.
So, to associate this special processing three edit controls to use functions could be ...

Call w32EditENTEREx (1, ...
Call w32EditENTEREx (2, ...

Call w32EditENTEREx (3 ...

Note 1: This function replaces the function group and w32EditENTER w32EditENTERn, although its use is not inconsistent set

Note 2: Although the most obvious use of this feature is to edit controls can also be used with list controls (List Box)

Ejemplo:

```
$SUB dlg1Event
SELECT CET_EVTYPE%
REM     Event notification save area
REM@# evtstart
CASE 15
        Rem Eventos Especiales añadidos en tiempo de ejecución
        SELECT CMDID%
        CASE 100
                SELECT tecla100%
                CASE 112
                     Rem Tratar la pulsacion de la tecla F1

                CEND

        CASE 101
                SELECT tecla101%
                CASE  33
                     Rem Re Pág.
                CASE 34
                     Rem Av Pág.
                CEND
        CEND
REM@# evtend
CEND
$EXIT


$SUB dlg1Init
REM     Initialize dialog save area
REM@# inistart
Rem poner el foco en un boton cualquiera  (200 p.ej) para hallar la referencia al cuadro de
dialogo.
Call cSetFocus (dlg1HDLG%,200)

Rem Obtener la referencia al dialogo. Solo es necesario llamar a esta funcion una vez al
inicio del dialogo.
call w32GetParent (addrof (padre%))
...

Rem Obtener la referencia al control de edición con ID 100
Call w32GetWindow (padre%,100,addrof (ed100%))
Rem Obtener la referencia al control de edición con ID 101
Call w32GetWindow (padre%,101,addrof (ed101%))

...

Rem Proceso de enter para id 100 y 101
call w32EditENTEREx (1,ed100%,100,addrof (tecla100%))
call w32EditENTEREx (2,ed101%,101,addrof (tecla101%)

REM@# iniend
```

call w32ListGetEventsExn (padre%,lis%,ID%,addrof (evento%),addrof (param1%),addrof (param2%))

Where n is a value from 1 to 9. Calling this function group can point to the system controls whose list of events generated by the user want to be notified.
The function allows you to associate an ID% to list events to distinguish this control has occurred. The last three parameters indicate the names of the variables that will hold at each instant information on the type of event occurred and its properties.

This instruction implements the same functionality as w32ListGetEventsn, except that the event notification is done differently to prevent some unwanted effects of change in focus that occurred sometimes when you used

the previous version. Requires 9.32ma version from which the runtime system to process the event CETW32 special number 15 by which it can detect additional notifications. The value of the variable reserved CMDID% contain the ID of the control that generated the event.

The interpretation of the parameters is as follows:

Father%, referring to the dialog box. (The value returned by the call to w32GetParent).
lis% control identifier list. (The value returned by the call to w32CreateListH).
ID%, value that designates the list ID.
Event% variable name that will contain the value of the number of events produced at each instant.
Param1%, the first property of the event. It returns -1 if not applicable.
Param2%, the second property of the event. It returns -1 if not applicable.

The information available for each type of event is as follows:

Evento	*Descripción*	*Propiedad 1*	*Propiedad 2*
1	Click con el botón izq. del ratón	Item (fila) de la lista	Sub-item (columna)
2	Cambio de estado (Selección o Check Box)	Item (fila) de la lista	-1
3	Doble click con el botón izq. del ratón	Item (fila) de la lista	Sub-item (columna)
4	Click con el botón derecho del ratón	Item (fila) de la lista	Sub-item (columna)
5	Pulsación de una tecla	Valor (ver w32Skeybd_event_down)	-1
6	Click en el título de una columna	-1	Columna de la lista

The usual place where to place the call to this function is the end of the section $ SUB Init.

In all dialogues that make up an executable should not be called more than once to the same function in this group. Thus, to define the event processing list controls three functions to be used...

Call w32ListGetEventsEx1...
Call w32ListGetEventsEx2...
Call w32ListGetEventsEx3...

Este grupo de funciones reemplaza al grupo de funciones w32SetEvents*n* y w32ListGetEvents*n*

Ejemplo:

```
$SUB dlg1Event
SELECT CET_EVTYPE%
REM     Event notification save area
REM@# evtstart
CASE 15
SELECT CMDID%
...
CASE 400
Rem 400 es el ID asociado a la lista en la llamada a w32ListGetEventsEx1
        Rem Notificaciones para la lista lis%
        Rem evento%->Numero de evento
        Rem l_item%->Propiedad 1
        Rem l_sub%->Propiedad 2
        SELECT evento%
        CASE 1
                Rem Click en un campo
                rem Msgbox "Click en el campo: "&str$ (l_item%)&","&str$ (l_sub%)

        CASE 2
                Rem Cambio de seleccion, o bien cambio de casilla de verificacion
                rem Msgbox "Ha cambiado el item: "&str$ (l_item%)
                call w32ListGetSelItem (lis%,addrof (item%))
                if item%<>-1
                then
                call w32ListGetText (lis%,item%,0,addrof (text$),addrof (ret%))
                call cSetCtrlValue (dlg1HDLG%,100,text$)
                ifend
```

```
        CASE 3
                Rem Doble Click en un campo
                Msgbox "Doble Click en el campo: "&str$ (l_item%)&","&str$ (l_sub%)

        CASE 4
                Rem Click con el botón de derecho en un campo
                Msgbox "Click con botón derecho en el campo: "&str$ (l_item%)&","&str$
(l_sub%)

        CASE 5
                Rem Se ha pulsado la tecla:
                Msgbox "Tecla: "&str$ (l_item%)

        CASE 6

                Rem Click en la cabecera de la columna
                Msgbox "Click en columna: "&str$ (l_sub%)
                call w32ListSortItems (lis%,l_sub%,0,1,addrof (ret%))

        CEND

...
CEND
REM@# evtend
CEND
$EXIT



$SUB dlg1Init
REM     Initialize dialog save area
REM@# inistart

call w32GetParent (addrof (padre%))
...
call w32CreateListH (padre%,10,195,475,180,addrof (lis%))
...
call w32ListGetEventsEx1 (padre%,lis%,400,addrof (evento%),addrof (l_item%),addrof
(l_sub%))
...
REM@# iniend
```

call w32TreeGetEventsExn (padre%,tree%,ID%,addrof (etree%),addrof (ptree%))

Where n is a value from 1 to 9. Calling this function group can point to the tree controls system on which events generated by the user want to be notified.
The function allows you to associate an ID% to list events to distinguish this control has occurred. The last two parameters indicate the names of the variables that will hold at each instant information on the type of event occurred and its properties.

This instruction implements the same functionality as w32TreeGetEventsn, except that the event notification is done differently to prevent some unwanted effects of change in focus that occurred sometimes when you used the previous version. Requires 9.32ma version from which the runtime system to process the event CETW32 special number 15 by which it can detect additional notifications. The value of the variable reserved CMDID% contain the ID of the control that generated the event.

The interpretation of the parameters is as follows:

Father%, referring to the dialog box. (The value returned by the call to w32GetParent).
tree% control identifier tree. (The value returned by the call to w32CreateTreeH).
ID%, which means the tree ID.
etree% variable name that will contain the value of the number of events produced at each instant.
ptree% ownership of the event.

The information available for each type of event is as follows:

Evento	*Descripción*	*Propiedad*
1	Click con el botón izq. del ratón	Referencia al elemento
2	Cambio de estado (Selección)	Referencia al elemento
3	Doble click con el botón izq. del ratón	Referencia al elemento
4	Click con el botón derecho del ratón	Referencia al elemento

5	Pulsación de una tecla	Valor (ver w32Skeybd_event_down)
6	Expansión de una rama	Referencia al elemento

The usual place where to place the call to this function is the end of the section $ SUB Init.

In all dialogues that make up an executable should not be called more than once to the same function in this group. Thus, to define the event processing tree controls three functions to be used ...

Call w32TreeGetEventsEx1 ...
Call w32TreeGetEventsEx2 ...
Call w32TreeGetEventsEx3 ...

This function group function group replaces w32TreeGetEventsn, although its use is not inconsistent set.

w32TreeSetImage call (tree% ref% indice_ima% addrof (ret%))

Sets the image tree item referenced by ref% with the image of the tree list on index% having índice_ima% (starting from 0).

w32ShellExecute call (father%, $ operation, $ file, $ arg, $ dir, mode% addrof (return%))

Performs an operation on the specified file.

The interpretation of the parameters is as follows:
Father%, parent window handle to receive any message box regarding implementation. If used, must be the value returned by the function w32GetParent otherwise specify 0.

$ Operation, one of the following values:

edit (open a document for editing)
explore (Explore the specified folder)
open (Opens the specified document)
print (Print the document specified)

$ file, file on which the operation is performed

$ arg if the specified file is the name of an executable, this parameter indicates the parameters that are passed to the application. Otherwise specify "".

Mode%, which indicates the way in which to display the window in accordance with the following table:

Hidden Window 0
Activates and displays the window 1
Activates the window and minimizes 2
Activates the window and maximizes 3
Activates the window in its most recent size and position 4
Activates the window and displays it in its current size and position 5
Minimizes the window and activates the appropriate following six
Minimizes the window that was active window remains 7
Displays the window in its current state remains active window 8
Active and restore a minimized window when this was 9

w32ShellExecuteEx call (father%, $ operation, $ file, $ arg, $ dir, mode% seg_timeout% addrof (return%))

Performs an operation on the specified file, expecting to conclude

The interpretation of the parameters is as follows:

Father%, parent window handle to receive any message box regarding implementation. If used, must be the value returned by the function w32GetParent otherwise specify 0.

$ Operation, one of the following values:

edit (open a document for editing)
explore (Explore the specified folder)
open (Opens the specified document)
print (Print the document specified)

$ file, file on which the operation is performed

$ arg if the specified file is the name of an executable, this parameter indicates the parameters that are passed to the application. Otherwise specify "".

Mode%, which indicates the way in which to display the window in accordance with the following table:

Hidden Window 0
Activates and displays the window 1
Activates the window and minimizes 2
Activates the window and maximizes 3
Activates the window in its most recent size and position 4
Activates the window and displays it in its current size and position 5
Minimizes the window and activates the appropriate following six
Minimizes the window that was active window remains 7
Displays the window in its current state remains active window 8
Active and restore a minimized window when this was 9

Seg_timeout%, maximum number of seconds that will be expected to finalize the implementation of the action.

Example:
w32ShellExecuteEx call (0, "open", "printpdf.vbs", "c: \ esptxt.pdf", "", 0.20, addrof (return%))

This call executes the script printpdf.vbs passing as argument the file name, and indicating a maximum wait time for the completion of the execution of 20 seconds.

(The script attached printpdf.vbs throws the background printing pdf document that is passed as a parameter, which can be expected to determine the status of the print and then close Adobe Reader. The return value is 1 if the file began to be printed, or 0 otherwise)

w32SetEnv call ($ var_name, $ variable-value, addrof (return%))

Sets the value of the environment variable indicating system as the first argument. In the event that the variable already contains a value added is indicated as second argument. This function can also be used to create own system variables. The changes will be reflected instantly on the Environment Variables section of the System option on the control panel.

The return value is 0 if the operation was successful or not 0 if an error occurred.

Requires Seller CETEnv.dll dated 19/06/2007 or later.

w32GetEnv call ($ var_name, addrof ($ value))

Returns the value of the environment variable indicating system as the first argument. In the event that there is no return the string "".

Requires Seller CETEnv.dll dated 19/06/2007 or later.

w32FtpGet call (sServidor $, $ sUser, sClave $, $ sPath, $ sDestino, addrof (return%))

This feature allows you to download a file from a server using the FTP protocol. Requires CETFTP.DLL have the file in the working directory.

```
Rem El archivo CETFTP.DLL debe estar en el directorio de los ejecutables
retorno%=0
print crt$("clear")

sServidor$="ftp.omega-st.es"
sUsuario$="seba@omega-st.es"
sClave$="omega"
sRuta$="/var/www/html/cosas/LigaF7b.pdf"
call Bgetcwd(addrof(sDonde$))
sDestino$=sDonde$ & "\LigaF7b.pdf"

print at$(1,1);"Descargando el archivo " & sRuta$
print at$(1,2); "En " & sDestino$

call w32FtpGet(sServidor$, sUsuario$, sClave$, sRuta$, sDestino$, addrof(retorno%))
Msgbox "Concluyo con retorno: "&str$(retorno%)
```

call w32FtpPut(sServidor$, sUsuario$, sClave$, sDestino$, sRuta$, addrof(retorno%))

This feature allows you to upload a file from a local medium to an FTP server. Requires CETFTP.DLL have the file in the working directory.

```
Rem CETFTP.DLL File must be in the directory of the executable
Return% = 0
crt print $ ("clear")

print at $ (1,1); "gordo.jpg Sending file ..."

sServidor $ = "ftp.omega-st.es"
sUser $ = "seba@omega-st.es"
sClave $ = "omega"
Bgetcwd call (addrof (sDonde $))
sDonde sPath $ = $ & "\ asd.pdf"
sDestino $ = "/ var / www / html / things / asd.pdf"

print at $ (1,1); "Sending file" & sPath $
print at $ (1,2); "In" & $ sDestino

w32FtpPut call ($ sServidor, sUser $, $ sClave, sDestino $, $ sPath, addrof (return%))
Msgbox "conclude with return:" & str $ (return%)


call MATADDROF (listatamanos $), addrof (return%))
```

w32FtpFilesInfo(server$, user$, password$, path$, MATADDROF(files$), MATADDROF(dates$), MATADDROF(sizes$), addrof(retorno%))

Gets an array with the list of files and subdirectories (preceded by the string "* D *") belonging to a particular directory to an FTP site. Besides other two arrays are returned with dates (in format YYYY-MM-DD HH: mm: ss) and sizes in bytes of the files. Dll is required cetftp dated 22/06/2010 or later.
The site's root directory must be specified with the string "/". Arrays that are passed as arguments must be sized previously.
The return value can be:

1 Operation completed successfully.
-1 Error in operation.

Ejemplo:

```
retorno%=0
dim archivos$(500)
dim fechas$(500)
dim tamanos$(500)
call
w32FtpFilesInfo("ftp.ya.com","jobrodriguez@ya.com","clave","/",MATADDROF(archivos$),MATADDR
OF(fechas$), MATADDROF(tamanos$), addrof(retorno%))
i=1
while archivos$(i)<>""

        print at(1,i);archivos$(i);" ";fechas$(i);" ";tamanos(i);" bytes"
i=i+1
wend
```

```
Msgbox "Concluyo con retorno: "&str$(retorno%)
```

call w32ImageBackgroundEx5(padre%, imagen$, ANCHO%, ALTO%, addrof(tapiz%))

Sets the background of a FormView or dialogue as image.
Requires that the file is in the directory CETExt.dll work.

```
cSetFocus call (dlg1HDLG%, 200)
w32GetParent call (addrof (parent%))
w32ImageBackgroundEx5 call (father% ". \ fondo.jpg", 800.600, addrof (Skin%))
```

w32KillProcess call ($ nombre_del_proceso, addrof (return%))

Ends the process indicated as the first argument. The name to be indicated is shown as image name in the Processes tab of Windows Task Manager. (For example: WINWORD.EXE EXCEL.EXE or)

The return value is 1 if it has managed to abort or 0 if the process does not exist.

Requires Seller CETProcess.dll dated 17/06/2007 or later.
Example:

```
call cForceExit
w32KillProcess call ("EXCEL.EXE" addrof (return%))
If return% = 1
Print "The process has been aborted"
else
Print "The process does not exist"
IfEnd
sleep 2
```

w32ImageBackgroundExn call (father%, $ image, width%, height%, addrof (Skin%))

Where n is a value from 1 to 5. Calling this function group to define the background wallpaper that displays a dialog box or a Form View. In all cases, the controls will be transparently overlaid on the carpet. Requires CETExt.dll file dated 09/12/2007 or later and CETPNG.dll file dated 19/04/2009 or later.

The interpretation of the parameters is as follows:

Father%, reference to the dialog or Form View. (Value returned by the function w32GetParent)
$ Path image file to be displayed as wallpaper. (It may be of type bmp, gif, jpg, etc. ..)
% W width in pixels of the image.
Alto high% pixels of the image in bmp.
Skin% is the identifier of the resulting image of the tapestry.

The usual place where to place the call to this function is the end of the section $ SUB Init.

In all dialogues that make up an executable should not be called more than once to the same function in this group. Thus, to define the three dialogues Skin functions to be used ...

Call w32ImageBackgroundEx1 ...
Call w32ImageBackgroundEx2 ...
Call w32ImageBackgroundEx3 ...

w32InitCETPNG call (father%)

Initializes the runtime to dynamically create controls transparent PNG32 images in the application. To display the images, you must set the background wallpaper, or FormView corresponding dialogue through the call to the function group w32ImageBackgroundExn dated 26/04/2009 or later.

The interpretation of the parameters is as follows:

Father%, reference to the dialog or Form View. (Value returned by the function w32GetParent)

Creating PNG images controls should be performed using the function w32CreateWindowH and indicating the value "CETPNG" to the argument for the type of control.

w32CreateWindowH call (father% ID%, $ type, style% estiloex%, title $, x%, y%, width%, height%, addrof (control%))

This feature allows you to create controls at runtime WINDOWS various types.

The meaning of the parameters is as follows:
Father%, reference to the dialog must be the value returned by w32GetParent.
ID%, Number to be used to identify the control to be created. It's the same identify using standard controls W32 CET.
Type $, is a string that identifies the type of control to create. Some examples: "CETPNG", "Edit", "Button", "SysDateTimePick32". (See appendix to this function)
Style% Determines the style that will be in control. Usually contains 1 or more constants specified below to which is applied the OR operator.

EstiloEx% Determines the extended style that will be in control. Usually contains 1 or more constants specified below to which is applied the OR operator.

$ Title, is the initial value of text to display the control (if applicable).
Control%, reference to the control that was created.

Example:
To create a control that displays a PNG on a background wallpaper could use the following code.

```
call w32CreateWindowH(dialogo%,idcontrol%,"CETPNG",WS_BASE% OR SS_NOTIFY% OR
SS_OWNERDRAW%,0,rutaarchivoPNG$,posx%,posy%,anchoimagen%,altoimagen%,addrof(controlPNG%))
```

Nota: Para recibir notificaciones de las pulsaciones del ratón debemos asignar como idcontrol% un valor, que no esté usado por otro botón, comprendido entre 200 y 299.

```
Anexo:
Rem Estilos de Ventana
WS_OVERLAPPED% = 00000000H
WS_POPUP% = 80000000H
WS_CHILD% = 40000000H
WS_CLIPSIBLINGS% = 04000000H
WS_CLIPCHILDREN% = 02000000H
WS_VISIBLE% = 10000000H
WS_DISABLED% = 08000000H
WS_MINIMIZE% = 20000000H
WS_MAXIMIZE% = 01000000H
WS_CAPTION% = 00C00000H
WS_BORDER% = 00800000H
WS_DLGFRAME% = 00400000H
WS_VSCROLL% = 00200000H
WS_HSCROLL% = 00100000H
WS_SYSMENU% = 00080000H
WS_THICKFRAME% = 00040000H
WS_MINIMIZEBOX% = 00020000H
WS_MAXIMIZEBOX% = 00010000H
WS_GROUP% = 00020000H
WS_TABSTOP% = 00010000H
WS_BASE%=(WS_CHILD% OR WS_VISIBLE%)

Rem Estilos extendidos
WS_EX_TRANSPARENT% = 00000020H
WS_EX_WINDOWEDGE% = 00000100H
WS_EX_CLIENTEDGE% = 00000200H
WS_EX_LAYERED% = 00080000H
WS_EX_NOPARENTNOTIFY% = 00000004H

Rem Controles estaticos
SS_LEFT% = 00000000H
SS_CENTER% = 00000001H
SS_RIGHT% = 00000002H
SS_ICON% = 00000003H
SS_BLACKRECT% = 00000004H
SS_GRAYRECT% = 00000005H
SS_WHITERECT% = 00000006H
SS_BLACKFRAME% = 00000007H
```

```
SS_GRAYFRAME% =  00000008H
SS_WHITEFRAME% = 00000009H
SS_SIMPLE% = 0000000BH
SS_LEFTNOWORDWRAP% = 0000000CH
SS_NOPREFIX% = 00000080H
SS_NOTIFY% =00000100H
SS_OWNERDRAW%=0000000DH

Rem Botones
BS_PUSHBUTTON% = 00000000H
BS_DEFPUSHBUTTON% = 00000001H
BS_CHECKBOX% = 00000002H
BS_AUTOCHECKBOX% = 00000003H
BS_RADIOBUTTON% = 00000004H
BS_3STATE% = 00000005H
BS_AUTO3STATE% = 00000006H
BS_GROUPBOX% = 00000007H
BS_USERBUTTON% = 00000008H
BS_AUTORADIOBUTTON% = 00000009H
BS_OWNERDRAW% = 0000000BH
BS_LEFTTEXT% = 00000020H
BS_MULTILINE%= 00002000H
BS_VCENTER% =  00000C00H

Rem Controles de Edicion
ES_LEFT% = 00000000H
ES_CENTER% = 00000001H
ES_RIGHT% = 00000002H
ES_MULTILINE% = 00000004H
ES_UPPERCASE% = 00000008H
ES_LOWERCASE% = 00000010H
ES_PASSWORD% = 00000020H
ES_AUTOVSCROLL% = 00000040H
ES_AUTOHSCROLL% = 00000080H
ES_NOHIDESEL% = 00000100H
ES_OEMCONVERT% = 00000400H
ES_READONLY% = 00000800H
ES_WANTRETURN% = 00001000H
ES_NUMBER% = 00002000H

Rem Barra de Scroll
SBS_HORZ% = 0000H
SBS_VERT% = 0001H
SBS_TOPALIGN% = 0002H
SBS_LEFTALIGN% = 0002H
SBS_BOTTOMALIGN% = 0004H
SBS_RIGHTALIGN% = 0004H
SBS_SIZEBOXTOPLEFTALIGN% = 0002H
SBS_SIZEBOXBOTTOMRIGHTALIGN% = 0004H
SBS_SIZEBOX% = 0008H

Rem List Box
LBS_NOTIFY% = 0001H
LBS_SORT% = 0002H
LBS_NOREDRAW% = 0004H
LBS_MULTIPLESEL% = 0008H
LBS_OWNERDRAWFIXED% = 0010H
LBS_OWNERDRAWVARIABLE% = 0020H
LBS_HASSTRINGS% = 0040H
LBS_USETABSTOPS% = 0080H
LBS_NOINTEGRALHEIGHT% = 0100H
LBS_MULTICOLUMN% = 0200H
LBS_WANTKEYBOARDINPUT% = 0400H
LBS_EXTENDEDSEL% = 0800H
LBS_DISABLENOSCROLL% = 1000H
LBS_NODATA% = 2000H
LBS_NOSEL% = 4000H
LBS_STANDARD% = (LBS_NOTIFY% OR WS_VSCROLL% OR WS_BORDER% OR WS_HSCROLL% OR
LBS_USETABSTOPS%)
```

call w32MAPISendMailMultipleFiles (tema$, contenido$, direccion_destino$, nombre_destino$, numero_ficheros%, mataddrof(ficheros$), addrof(ret%))

Enables sending email without user intervention attaching many files as desired.

The syntax is:
$ Subject, Subject of e-mail
$ Content, email content
$ Direccion_detino, e-mail address, beginning with SMTP (eg SMTP: job@omegasystems.com)
$ Nombre_destino, Alternative name to specify the person for whom the e-mail

numero_ficheros%, number of files attached. It can be 0 or more.
$ files, matrix files with the relationship that you want to attach.
ret% return value. A value other than 0 indicates error.

Example:
```
dim files $ (2)
files $ (1) = "c: \ cetbin \ Brestore.log"
files $ (2) = "c: \ cetbin \ install.log"
w32MAPISendMailMultipleFiles call ("Log Files", "Attachment 2 log files", "SMTP:
job@omegasystems.com", "Job Rodriguez", 2, mataddrof ($ file), addrof (ret%))
msgbox str $ (ret%)
```

w32SetCtrlColorStatic call (father% rt% vt%, at% rf% vf% af%)

The call to this function allows you to set the text color and background of all static texts (text, radio button, check box and group boxes) of a dialogue

The interpretation of the parameters is as follows:

Father%, reference to the dialog. (Value returned by the function w32GetParent)
rt% value of R (red) text color; Valid values are 0 .. 255. Use the value -1 to indicate that you want to use the default color for that control.
vt% component value V (green) text color; Valid values are 0 .. 255. If parameter rt% -1 value is specified, in this you can specify the value returned by w32ImageBackgroundn if you want the background color is the same as the tapestry of dialogue.
at% value component (blue) text color; Valid values are 0 .. 255
rf% value of R (red) the background color of the text; Valid values are 0 .. 255. Use the value -1 to indicate that you want to use a transparent background for that control.
vf% component value V (green) background color of the text; Valid values are 0 .. 255
af% value component (blue) background color of the text; Valid values are 0 .. 255

The usual place where to place the call to this function is the end of the section $ SUB Init.

Example: see dialogue project

w32SetCtrlColorEdit call (father% rt% vt%, at% rf% vf% af%)

The call to this function allows you to set the text color and background checks of all dialogue editing

The interpretation of the parameters is as follows:

Father%, reference to the dialog. (Value returned by the function w32GetParent)

rt% value of R (red) text color; Valid values are 0 .. 255. Use the value -1 to indicate that you want to use the default color for that control.
vt% component value V (green) text color; Valid values are 0 .. 255. If parameter rt% -1 value is specified, in this you can specify the value returned by w32ImageBackgroundn if you want the background color is the same as the tapestry of dialogue.
at% value component (blue) text color; Valid values are 0 .. 255

rf% value of R (red) the background color of the text; Valid values are 0 .. 255. Use the value -1 to indicate that you want to use a transparent background for that control.
vf% component value V (green) background color of the text; Valid values are 0 .. 255
af% value component (blue) background color of the text; Valid values are 0 .. 255

The usual place where to place the call to this function is the end of the section $ SUB Init.

Example: see dialogue project

w32SetCtrlColorListBox call (father% rt% vt%, at% rf% vf% af%)

The call to this function allows you to set the text color and background checks of all dialogue list.

The interpretation of the parameters is as follows:

Father%, reference to the dialog. (Value returned by the function w32GetParent)
rt% value of R (red) text color; Valid values are 0 .. 255. Use the value -1 to indicate that you want to use the default color for that control.
vt% component value V (green) text color; Valid values are 0 .. 255. If parameter rt% -1 value is specified, in this you can specify the value returned by w32ImageBackgroundn if you want the background color is the same as the tapestry of dialogue.
at% value component (blue) text color; Valid values are 0 .. 255
rf% value of R (red) the background color of the text; Valid values are 0 .. 255. Use the value -1 to indicate that you want to use a transparent background for that control.
vf% component value V (green) background color of the text; Valid values are 0 .. 255
af% value component (blue) background color of the text; Valid values are 0 .. 255

The usual place where to place the call to this function is the end of the section $ SUB Init.

Example: see dialogue project

w32ListSetSubItemImage call (lis% item% column%, indice_ima% addrof (ret%))

Sets the image of the cell defined by the row and column item%% column list belonging to the image lis% with the associated list that has index lis% índice_ima% (starting from 0).

w32CreateImageList call (W%, $ file, addrof (ima%))

Create a reference to a file. Bmp must contain an image list.
The width of each individual image in pixels should be specified as the first parameter.
It returns a reference to the list of images in the last argument. The call
This function has no visual effect, only creates a reference to a list
of images that will then be referenced by its position in the list controls and
tree.

w32ListImageList call (lis%, ima% addrof (ret%))

Associated with the list lis% image list referenced by ima%. In most cases only have to make the call to this function only once per list.

w32BLook call (index% button% $ rutaimagen_on, rutaimagen_off $, $ rutaimagen_push, width%, height%, addrof (imagen_on%), addrof (imagen_off%), addrof (imagen_push%))

The call to this function allows you to associate different images to a button for each of the following situations:

• The mouse is positioned over the button area (imagen_on)
• The mouse is not positioned over the button area (imagen_off)
• The user has pressed the button (imagen_push)

Requires CETExt.dll file dated 09/12/2007 or later.

The interpretation of the parameters is as follows:

index% integer value between 1 and 50
button%, reference to the button. (Value obtained with w32GetWindow)
$ rutaimagen_on, Route of the image to display when the mouse is over the button. It can be any graphic file: jpg, gif, bmp, ...
$ rutaimagen_off, Route of the image to display when the mouse is not over the button. It can be any graphic file: jpg, gif, bmp, ...
$ rutaimagen_push, Route of the image to be displayed when the user hold the button. It can be any graphic file: jpg, gif, bmp, ...

width% Width in pixels of the images to be attached to the button.
high%, height in pixels of the images to be attached to the button.
imagen_on%, reference to the associated image when the mouse is over the button. This reference allows release resources occupied by the image w32DeleteObject invoking function.
imagen_off%, reference to the associated image when the mouse is not over the button. This reference allows release resources occupied by the image w32DeleteObject invoking function.
imagen_push%, reference to the associated image when the user holds down the button. This reference allows release resources occupied by the image w32DeleteObject invoking function.

In all dialogues that make up an executable to use an index value% different for each button.

So, to associate this special processing features three buttons to use could be ...

Call w32BLook (1, ...
Call w32BLook (2, ...
Call w32BLook (3, ...

w32DownloadWebFile call ($ dominio_servidor_web, ruta_completa_archivo_web $, $ ruta_archivo_local, addrof (return%))

Download any file hosted on the web and store it on the local file system. Requires CETFTP.dll dll dated 27/12/2009 or later.

The interpretation of the parameters is as follows:

$ Dominio_servidor_web, full web address for the web server domain. (For example: http://www.omegasystems.com)
$ Ruta_completa_archivo_web, full path of the file you want to download web. (For example: http://www.omegasystems.com/cet_descargas/servidor.zip)
$ Ruta_archivo_local, full path of the local file to be created.
Return%, a value of -1 indicates that an error has occurred.

w32FtpCreateDirectory call (FTPSite $, $ user, $ key, $ path, $ nuevodir, addrof (return%))

Create an FTP directory with the name indicated as 5 argument from the route indicated as 4 argument.
Dll is required cetftp dated 15/06/2008 or later.

The return value can be:
1 Operation completed successfully.
-1 Error in the connection to the ftp site
-2 Error in locating the route.
-3 Error creating directory. (May already exist)

Example:

```
call w32FtpCreateDirectory ("ftp.ya.com", "jobrodriguez@ya.com", "clave", "/", "omega",
addrof(retorno%))
call w32FtpCreateDirectory("ftp.ya.com", "jobrodriguez@ya.com", "clave", "/omega",
"uploads", addrof(retorno%))
```

call w32ListGetEventsExn (padre%, lis%, ID%, addrof(evento%), addrof(param1%), addrof(param2%))

Where n is a value from 1 to 9. Calling this function group can point to the system controls whose list of events generated by the user want to be notified.
The function allows you to associate an ID% to list events to distinguish this control has occurred. The last three parameters indicate the names of the variables that will hold at each instant information on the type of event occurred and its properties.

This instruction implements the same functionality as w32ListGetEventsn, except that the event notification is done differently to prevent some unwanted effects of change in focus that occurred sometimes when you used the previous version. Requires 9.32ma version from which the runtime system to process the event CETW32

special number 15 by which it can detect additional notifications. The value of the variable reserved CMDID% contain the ID of the control that generated the event.

The interpretation of the parameters is as follows:

Father%, reference to the dialog box. (The value returned by the call to w32GetParent).
lis% control identifier list. (The value returned by the call to w32CreateListH).
ID%, which means the ID of the list.
Event%, the variable name that will contain the value of the number of events produced at each instant.
Param1%, First owned event. It returns -1 if not applicable.
Param2% Second Event property. It returns -1 if not applicable.

The information available for each type of event is as follows:

Evento	*Descripción*	*Propiedad 1*	*Propiedad 2*
1	Click con el botón izq. del ratón	Item (fila) de la lista	Sub-item (columna)
2	Cambio de estado (Selección o Check Box)	Item (fila) de la lista	-1
3	Doble click con el botón izq. del ratón	Item (fila) de la lista	Sub-item (columna)
4	Click con el botón derecho del ratón	Item (fila) de la lista	Sub-item (columna)
5	Pulsación de una tecla	Valor (ver w32Skeybd_event_down)	-1
6	Click en el título de una columna	-1	Columna de la lista

The usual place where to place the call to this function is the end of the section $ SUB Init.

In all dialogues that make up an executable should not be called more than once to the same function in this group. Thus, to define the event processing list controls three functions to be used ...

Call w32ListGetEventsEx1 ...
Call w32ListGetEventsEx2 ...
Call w32ListGetEventsEx3 ...

This function group function group replaces w32SetEventsn and w32ListGetEventsn
Ejemplo:

```
$SUB dlg1Event
SELECT CET_EVTYPE%
REM     Event notification save area
REM@# evtstart
CASE 15
SELECT CMDID%
...
CASE 400
Rem 400 es el ID asociado a la lista en la llamada a w32ListGetEventsEx1
        Rem Notificaciones para la lista lis%
        Rem evento%->Numero de evento
        Rem l_item%->Propiedad 1
        Rem l_sub%->Propiedad 2
        SELECT evento%
        CASE 1
                Rem Click en un campo
                Msgbox "Click en el campo: "&str$(l_item%)&","&str$(l_sub%)
        CASE 2
                Rem Cambio de seleccion, o bien cambio de casilla de verificacion
                rem Msgbox "Ha cambiado el item: "&str$(l_item%)
                call w32ListGetSelItem(lis%,addrof(item%))
                if item%<>-1
                then
                call w32ListGetText(lis%,item%,0,addrof(text$),addrof(ret%))
                call cSetCtrlValue(dlg1HDLG%,100,text$)
                ifend
        CASE 3
                Rem Doble Click en un campo
                Msgbox "Doble Click en el campo: "&str$(l_item%)&","&str$(l_sub%)
        CASE 4
                Rem Click con el botón de derecho en un campo
                Msgbox "Click con botón derecho en el campo:
"&str$(l_item%)&","&str$(l_sub%)

        CASE 5
                Rem Se ha pulsado la tecla:
                Msgbox "Tecla: "&str$(l_item%)
```

```
        CASE 6
                Rem Click en la cabecera de la columna
                Msgbox "Click en columna: "&str$(l_sub%)
                call w32ListSortItems(lis%,l_sub%,0,1,addrof(ret%))
        CEND
...
CEND
REM@# evtend
CEND
$EXIT
$SUB dlg1Init
REM     Initialize dialog save area
REM@# inistart

call w32GetParent(addrof(padre%))
...
call w32CreateListH(padre%,10,195,475,180,addrof(lis%))
...
call w32ListGetEventsEx1(padre%,lis%,400,addrof(evento%),addrof(l_item%),addrof(l_sub%))
...
REM@# iniend
```

call w32SetCtrlColorN

Set of functions that allow you to set the colors of both the FormView controls and controls in the Dialogues, where N is the group of color settings.

w32SetCtrlColor1 call (father% c500% -1,0,0,255,255,0)
order parameters
1 - Handle to the control container to modify.
2 - Control Manager to modify.
3 - Value 0-255 scale concerning Red for background, -1 for transparent background.
4 - Value 0-255 scale concerning Green for background.
5 - Value 0-255 scale concerning Blue for the background.
6 - value of 0 to 255 scale regarding Red for text.
7 - Value 0-255 scale concerning Green for text.
8 - Value 0-255 scale concerning Blue for text.

w32CreateWebCtrl call (father% idcontrol%, $ source, posx% posy%, width%, height%, addrof (control%))

This feature allows you to create controls at runtime hosting websites or files stored on disk

The significance of the main parameters are as follows:
Father%, reference to the dialog must be the value returned by w32GetParent.
Idcontrol%, Number to be used to identify the control to be created. It's the same identify using standard controls W32 CET.
$ Source, is a string that identifies the website (eg, www.google.es) or file (eg: c: \ manual.html)

The return value is the control ID or -1 if an error occurred.

Requires CetWeb.dll DLL dated 16/12/2009 or later.

w32SetScreenResolution call (width%, height%, color% addrof (return%))

This feature allows you to automatically change the screen resolution at runtime. The first and second argument must specify the horizontal and vertical resolution respectively. The third argument is the number of available colors measured in bits per pixel. This value depends on the model of computer video adapter and is recommended to use the value returned by the native function call cDisplayInfo CET. (See example ssr.b)
The return value is 1 if the operation was successful, -1 if reboot is required for the changes to take effect or -2 if the set of arguments indicated is incorrect.

This feature requires Seller CETExt.dll dated 09/12/2007 or later.

w32FtpDir call (FTPSite $, $ user, $ password, $ directory, MATADDROF (listaarchivos $), addrof (return%))

Gets an array with the list of files and subdirectories of a given directory belonging to an FTP site. Dll is required cetftp dated 12/11/2007 or later.
The site's root directory must be specified with the string "/". The array that is passed as an argument must be dimensioned previously. If the operation completes successfully the array will contain a list of files and subdirectories, the latter preceded by the string "* D *".

The return value can be:
1 Operation completed successfully.
-1 Error in operation.
Example:

```
dim result $ (500)
w32FtpDir call ("ftp.ya.com", "jobrodriguez@ya.com", "key", "/", MATADDROF ($ result),
addrof (return%))
i = 1
results while $ (i) <> ""
        print at (1, i) $ result (i)
i = i +1
wend
Msgbox "conclude with return:" & str $ (return%)
```

w32FtpDel call (FTPSite $, $ user, $ key, $ ficheroftp, addrof (return%))

Deletes a file from an FTP location. Dll is required cetftp dated 12/11/2007 or later.

The return value can be:
1 Operation completed successfully.
-1 Error in the connection to the ftp site
-2 Error processing file.

w32BLoadAnyImageH call (r200% "benefits_mac.JPG", 130, 120, addrof (Return%))

call w32USBDisable (addrof (return%))

Prevents the use of removable USB storage devices on the computer.
The return will be 1 is the operation was successful, or 0 otherwise.
Requires CETUSB.dll file dated 04/12/2005 or later.

w32USBEnable call (addrof (return%))

Allows use of USB removable storage devices on the computer.
The return will be 1 is the operation was successful, or 0 otherwise.
Requires CETUSB.dll file dated 04/12/2005 or later.

CETGRID.DLL

With the library version dated 15/08/05 CETGRID added the following functionality:
1) Extension of operating the ENTER key on the control grid when working with environment variable B_CREQTAB = 1, so that the behavior when pressing that key would be to focus on the relevant cell of the grid instead of passing the focus to the next control on the form or dialog.
2) Advanced Edition of the cells in the grid when you press the spacebar.
3) Correction of errors in the corresponding cell positioning by pressing ENTER.

Call w32GridSetColumnAlign (grid%, column% alignment%)

Align all cells in the grid column%% you specify. The third argument values may be:
0 Align Left
1 Alignment centered
2 Right alignment
Requires Seller CETGRID.DLL dated 8/15/05 or later.

w32IsEnabled call (control% addrof (enabled%))

Determines whether a control is enabled or not. The interpretation of the parameters is as follows:
Control% Control identifier. It must be the value returned by the function w32GetWindow

Enabled% Returns 1 if the control is enabled.

w32IsVisible call (control% addrof (visible%))

Determines whether a control is visible or not. The interpretation of the parameters is as follows:
Control% Control identifier. It must be the value returned by the function w32GetWindow
Visible% Returns 1 if the control is visible.

w32FolderSpace call ($ rutacarpteta, addrof ($ size), addrof (ret%))

Returns the total size of all files in a folder and all subfolders. (Requires CETExt.dll file dated 18/06/2005 or later)
Example:

```
call w32GetFolderSpace ("C: \ cetbin" addrof ($ size), addrof (ret%))
```

w32ListView call (father% nele%, $ options, $ title, $ message, $ icon, anchotexto% flags%, selec_unica% $ selpordefecto, addrof ($ selection), addrof (return%))

Function that can be invoked from a program in graphical mode or in text mode and that, with a single line of code, to create a visual list of format options (Grid lines, check boxes). The user can choose one or more of the elements that will be passed as arguments to the function in the string $ options with the delimiter character: |. Also you can specify the dimensions of the section of text that can be included in the dialogue, the default selections are punctuated dialogue and icon. The function will return the button pressed by the user as well as a string with a sequence of 1 and 0 to indicate the user's selections. It runs the following line of code until the input is complete. (Requires CETExt.dll file dated 18/06/2005 or later)

The interpretation of the parameters is as follows:

Father% Return value w32GetParent function, if text mode programs specify 0.
nele% Number of elements contained in the list of options. Maximum allowed 5000.
$ options, string with all the options that are displayed in the list separated by the delimiter character |
$ Title, Title of the window.
$ Message, Message to be presented to the user.
Icon% numeric value associated with the icon is displayed in the dialog. The possible values are:
16 shows the critical message icon.
32 shows the warning icon question.
48 shows the warning icon.
64 shows the information message icon.

anchotexto%, Width, in pixels, of the text to be displayed as a message to the user. This value also influences the width of the elements of the list.

Flags% numeric value associated with the visual style of the list. It can be one, or the sum of the following values:

1 Displays the list with gridlines
4 Displays a check box to the left of each item in the list.

Selec_unica%, will be worth 1 if allowing only choose one option from the list to the user. Will be 0 if it allows the user to select multiple options.

$ Selpordefecto, string of zeros and / or one to indicate the initial selection to be presented to the user. This string will contain many characters as there are elements in the list of options.

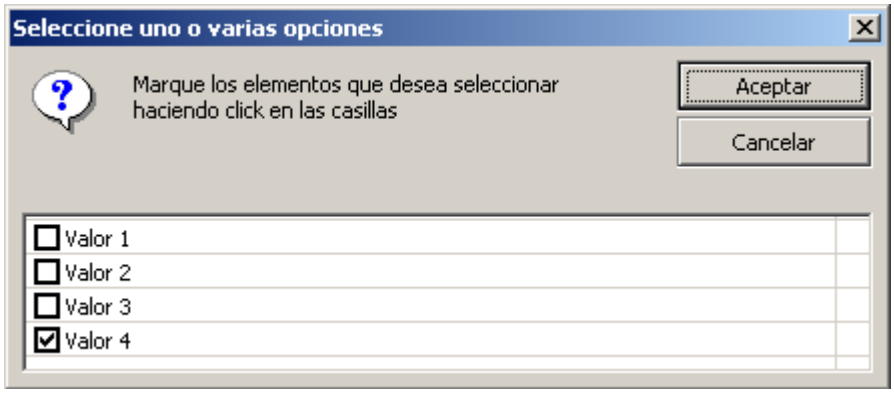
$ selection, string of zeros and ones to represent the final selection made by the user.

Return%, may have one of the following values indicating the button pressed by the user:
1 OK
2 Cancel

Example:

```
w32ListView call (0.4, "Value 1 | Value 2 | Value 3 | Value 4", "Select one or more
options", "Check the items you want to select" & chr $ (13) & "by clicking on the boxes" ,
+1.0 32,250,4, "0001", addrof ($ value), addrof (ret%))
```

The execution of this line of code displays the following dialog to the user:



call w32CheckBox (padre% , nele%, opciones$, titulo$, mensaje$, icono%, anchotexto%, selpordefecto$, addrof(seleccion$), addrof(retorno%))

Function that can be invoked from a program in graphical mode or in text mode, and with a single line of code creates a dialog box with multiple check-box controls. The user can choose one or more of the elements that will be passed as arguments to the function in the string $ options with the delimiter character: |. Also you can specify the dimensions of the section of text that can be included in the dialogue, the default selections are punctuated dialogue and icon. The function will return the button pressed by the user as well as a string with a sequence of 1 and 0 to indicate the user's selections. It runs the following line of code until the input is complete. (Requires CETExt.dll file dated 18/06/2005 or later)

The interpretation of the parameters is as follows:

Father% Return value w32GetParent function, if text mode programs specify 0.
nele%, Number of check-box controls. Maximum allowed 5000.
$ options, string with all the options that are displayed separated by the delimiter character |
$ Title, Title of the window.
$ Message, Message to be presented to the user.
Icon% numeric value associated with the icon is displayed in the dialog. The possible values are:

16 shows the critical message icon.
32 shows the warning icon question.
48 shows the warning icon.
64 shows the information message icon.

anchotexto%, Width, in pixels, of the text to be displayed as a message to the user.

$ Selpordefecto, string of zeros and / or one to indicate the initial selection to be presented to the user. This string will contain many characters as check-box controls are going to show.

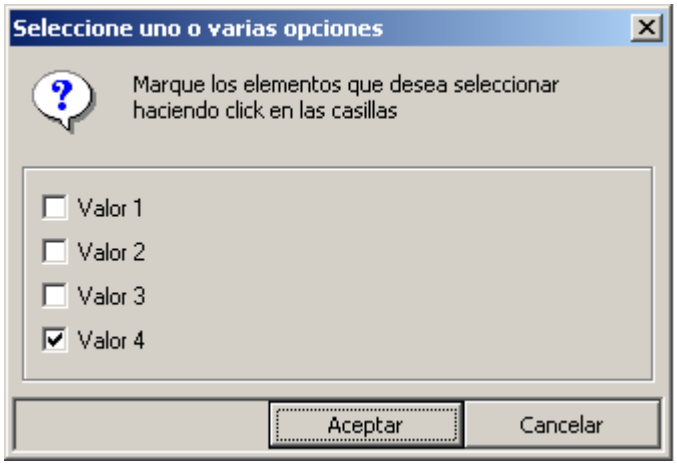
$ selection, string of zeros and ones to represent the final selection made by the user.

Return%, may have one of the following values indicating the button pressed by the user:
1 OK
2 Cancel

Example:

```
w32CheckBox call (0.4, "Value 1 | Value 2 | Value 3 | Value 4", "Select one or more
options", "Check the items you want to select" & chr $ (13) & "by clicking on the boxes" ,
32.250, "0001", addrof ($ value), addrof (ret%))
```

The execution of this line of code displays the following dialog to the user:



call w32GetDefaultPrinter (addrof(impresora$),addrof(ret%))

Gets the name of the default printer that has established the team. (Requires CETprint.dll file dated 18/04/2005 or later)

w32SetDefaultPrinter call ($ printer, addrof (ret%))

Set as default printer team which is indicated as first parameter. (Requires CETprint.dll file dated 18/4/2005 or later)

w32InputBox call (father%, $ message, $ title, style% $ valorinicial, addrof ($ value), addrof (return%))

Function dialog box which allows the user introduccir text then you get in return, as well as an indicator of the button you pressed. It runs the following line of code until the input is complete. (Requires CETExt.dll file dated 05/05/2005 or later)

The interpretation of the parameters is as follows:

Father% Return value w32GetParent function, if text mode programs specify 0.
$ Message, Message to be presented to the user.
$ Title, Title of the window.
Style% This parameter is equivalent to the function w32MessageBox. Numeric expression that is the sum of values specifying the number and type of buttons that are intended to display, the icon style to use and the identity of the default button.
The possible values are:
0 Display OK button only.
1 Display OK and Cancel buttons.
2 Displays the Abort, Retry, and Ignore.
3 shows the buttons Yes, No, and Cancel.
4 Display Yes and No buttons
5 Display Retry and Cancel buttons.

16 shows the critical message icon.
32 shows the warning icon question.
48 shows the warning icon.
64 shows the information message icon.

0 First button is default.
256 Second button is default.
512 Third button is default.

Valorinicial $, is the default value that appears in the text box.

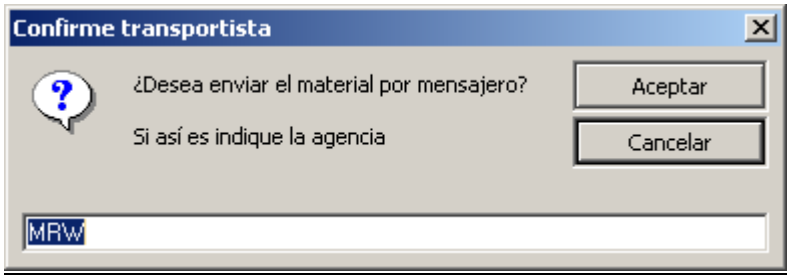
$ Value, Value finally written by the user.
Return%, may have one of the following values indicating the button pressed by the user:

1 OK
2 Cancel
3 Cancel
4 Retry
5 Ignore
6 Yes
7 No

Example:

```
w32InputBox call (0, "Do you want to send the material by courier?" & CHR $ (13) & CHR $
(13) & "If so explain the agency", "Confirm carrier", 1 +32 +256, "MRW" addrof ($ value),
addrof (ret%))
```

The execution of this line of code produces the following output:



call w32Splash (padre%, imagen$, mensaje$, posx%, posy%, fuente$, puntos%, r%, g%, b%, miliseg%, addrof(retorno%))

Allows a display screen (splash) application indicating the background image, text, style, color and location as well as the maximum time to be displayed. The return will be: 'timeout', or 'Click User'. (Requires CETExt.dll file dated 05/05/2005 or later)

The interpretation of the parameters is as follows:
Father% Return value w32GetParent function, if text mode programs specify 0.
$ image, image file path to be displayed.
$ Message, Text to be written on the image.
Posx%, horizontal position, in pixels, in which to display the text
Posy%, vertical position, in pixels, in which to display the text
$ Source, the source name to be used to render the text.
Points%, Font Size
r%, value of R (red) text color; Valid values are 0 .. 255
g%, component value G (green) text color; Valid values are 0 .. 255
b%, value of component B (blue) color of the text; Valid values are 0 .. 255
miliseg% Number of milliseconds to display the splash screen. When specifying 0, is displayed until the user clicks the mouse over it.
% Return, defaults to 1 if the user clicked on the screen or 64 if timed.

example:

```
call w32Splash(0,"csi.gif","Autorizado a: Asesoria Narbona"&chr$(13)&"Numero de serie: 123-
456789",130,180,"Tahoma",11,255,0,0,10000,addrof(ret%))
```

call w32GridSortItems (grid%, columna%, alfanum%, orden%, addrof(ret%))

Sorts the elements of a grid based on the values and types of the fields in the specified column.

grid%, at grid reference returned by the function w32CreateGrid
Column% Column grid to sort by. (Starting from 1)
alphanumeric%, 0 alphanumeric sort, 2 and 3 numeric sort sort by date.
order%, 1 up and 2 down.

Ejemplo:

Líneas de la Factura 08541

	Código	Descripción	Unid.	Precio	Fecha	Revisado
1	9687250	VARIO CASE P/SISTEMA VAS C/TAP	1	886.4	24-06-2005	☑
2	9314132	DESTORNILLADOR EXAGONAL"T"3.0	1	76.1	25-06-2005	☑
3	9388380	VAINA SUJECION DESTORNILLADOR	1	215.7	26-06-2005	☑
4	9388532	AVELLANADOR PARA USS VAS	1	171.7	27-06-2005	☑
5	9388357	EXTRACTOR PARA USS VAS	1	244	28-06-2005	☑
6	9497191	TORNILLO USS VAS 6.2X30 T.CPTO	2	378.8	29-06-2005	☑
7	9497192	TORNILLO USS VAS 6.2X35 TAN	4	378.8	30-06-2005	☑
8	9497193	TORNILLO USS VAS 6.2X40 TAN	4	378.8	01-07-2005	☑
9	9497194	TORNILLO USS VAS 6.2X45 TAN	4	378.8	02-07-2005	☑
10	9497195	TORNILLO USS VAS 6.2X50 TAN	2	378.8	03-07-2005	☑

For the grid of figure could use the following call management:

```
$SUB grid1Event
SELECT CET_EVTYPE%
REM     Event notification save area
REM@# evtstart
CASE 15
        Rem Eventos Especiales....
        SELECT CMDID%
        Rem 400 es el ID del control Grid% que le hemos asignado en w32CreateGrid
        CASE 400
                call w32GridGetEvent(grid%,addrof(ev%))

                SELECT ev%

                CASE 24
```

```
call w32GridGetEventP1(grid%,addrof(p1%))
call w32GridGetEventP2(grid%,addrof(p2%))
Rem Se ha hecho click en un encabezado linea superior)
if p2%<>0


        if p2%<3 or p2%>5 then al%=0
        if p2%=3 or p2%=4 then al%=2
        if p2%=5 then al%=3
        call w32GridSortItems(grid%,p2%,al%,1,addrof(ret%))

ifend
```

call w32GridGetEvent (grid%, addrof (evento%))

Returns the number of events that generated the grid, and w32GridEventP2 w32GridGetEventP1 functions should call ahead to get the value of the first and second property of the event that just occurred. Using CETGRID.dll version dated 11/06/2004 or later you can get a new notification that was added to detect keystrokes that combine with Control.

Evento	Descripción	Propiedad 1	Propiedad 2
33	Pulsación de Ctrl + Tecla	Valor de la tecla empezando por 1	0

Ejemplo:

```
$SUB grid1Event
SELECT CET_EVTYPE%
REM     Event notification save area
REM@# evtstart
CASE 15
        Rem Eventos Especiales....
        SELECT CMDID%
        Rem 400 es el ID del control Grid% que le hemos asignado en w32CreateGrid
        CASE 400
                call w32GridGetEvent(grid%,addrof(ev%))
                SELECT ev%
                CASE 33
                        call w32GridGetEventP1(grid%,addrof(p1%))
                        Msgbox "Ha pulsado (CONTROL + "&ucase$(chr$(64+p1%))&")"
```

call w32CreateStaticClassEx(padre%, addrof (evento%))

Create a new control class called "CETStaticLinkEx" for dialogue or Form View is passed as the first argument. Once the call can be created using the function controls w32CreateWindowH Link and specifying the type of control "CETStaticLinkEx". The controls show a hand like mouse pointer when the user is placed on them. (Cursors.dll Seller is required). When the user clicks on the control will generate a special event and the number 15 CMDID% reserved variable will take the value of the ID of the control that was specified as the second argument in the function w32CreateWindowH also variable specified as the second argument of this w32CreateStaticClassEx function worth -1.

w32StaticSetImageH call (control% $ image, 20, 20, addrof (Return%))

Associates a static image control or Class "CETStaticLink" so that instead of showing the text shows the $ image file that is indicated as second argument to the width and height in pixels indicated as third and fourth arguments respectively.

Ejemplo:

```
$SUB hot1Init
REM     Initialize dialog save area
REM@# inistart
#include "estilos.h"
Rem El fichero estilos contiene la definición de las constante empleadas en
w32CreateWindowH
.
.
.

Rem Permitir crear controles link en tiempo de ejecución
call w32CreateStaticClassEx(padre%,addrof(prop%))
```

```
Rem Crear un control CETStaticLinkEx al que se le puede asociar una imagen, esto se indica
por el flag SS_BITMAP

call w32CreateWindowH(padre%,521,"CETStaticLinkEx",WS_BASE% OR SS_NOTIFY% OR
SS_BITMAP%,0,"",110,46,20,20,addrof(r521%))

Rem Se le asocia la imagen al control que acaba de crearse.
call w32StaticSetImageH(r521%,".\imagen\abrir.bmp",20,20,addrof(ret%))
.
.
.

$SUB hot1Event
SELECT CET_EVTYPE%
REM     Event notification save area
REM@# evtstart
CASE 15
        SELECT CMDID%
        CASE 521
        Rem Eventos para el control Link con ID número 521
        if prop%=-1
                Rem Se ha hecho click en la etiqueta
        ifend
```

call w32BlockInput(bloquea%)

Disable or enable the keyboard and mouse as a function of the argument is 1 or 0. The only combination that is recognized Ctrl + Alt + Delete

w32GetAdaptersInfo call (num% addrof ($ name), addrof ($ desc), addrod ($ mac), addrof ($ ip), addrof ($ mask), addrof (ret%))

Returns information about the network adapters installed on the system. The interpretation of the parameters is as follows:

Num%, network adapter number on which you want information (starting from 0).
$ Name, Name which identifies the adapter WINDOWS
$ Desc, Model Description Network Adapter
$ Mac, Address MAC Address of the adapter.
$ IP, IP address associated with the adapter
$ Mask, Netmask
Ret%, 0 indicates error

w32MenuCheckRadioItem (menu% ini% end% sel% addrof (ret%))

Allows a group of mutually exclusive options menu and select one being marked with a radio button. The interpretation of the parameters is as follows:

Menu% is the menu handle. (Must be the value returned by w32CreateMenu)
Ini% menu item identifier initial clustering.
End% menu item identifier end of the pool.
Sel%, the group element to be selected. Must be a value between initial and end%%
Ret%, 0 indicates error.

w32MenuInsertItemEx call (menu% id%,% type, $ text, check% desh% addrof (ret%))

Add a menu item to the end of%. With the possibility of being checked at all or disabled.

The interpretation of the parameters is as follows:
Menu% is the identifier of the menu to which you add the item. (Must be the value returned by w32CreateMenu)
ID%, ID that is associated with the menu item. It is intended to help determine the selection made by the user. In case the input is in turn a submenu, the value of this parameter must be the same identifier.

Type%, is a value between 0 and 2, which determines the type of entry being inserted.
0, The item is a selectable item inserted normal.
1, The item to be inserted is itself another menu (submenu)

2, The item to be inserted is a separation line unselectable.

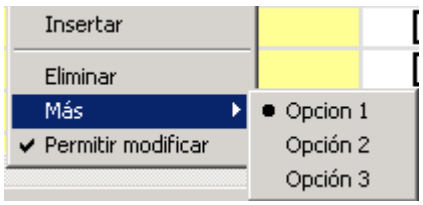
$ Text, Text string to be displayed in the entrance.

Check%, If 1 put a check box to the left of the item.
Desh%, If 1 disables the element.

Ejemplo:

```
Rem Creación del menu contextual
call w32CreateMenu(addrof(submenu%))
call w32MenuInsertItem(submenu%,6,0,"Opcion 1",addrof(ret%))
call w32MenuInsertItem(submenu%,7,0,"Opción 2",addrof(ret%))
call w32MenuInsertItem(submenu%,8,0,"Opción 3",addrof(ret%))

call w32CreateMenu(addrof(menu%))
call w32MenuInsertItem(menu%,1,0,"Insertar",addrof(ret%))
call w32MenuInsertItem(menu%,2,2," ",addrof(ret%))
call w32MenuInsertItem(menu%,3,0,"Eliminar",addrof(ret%))
call w32MenuInsertItem(menu%,submenu%,1,"Más",addrof(ret%))
call w32MenuInsertItemEx(menu%,4,0,"Permitir modificar",1,0,addrof(ret%))
call w32MenuCheckRadioItem(menu%,6,8,6,addrof(ret%))
```

Al mostrar el menu anterior con la función w32MenuPopup el resultado es el siguiente:



call w32EncryptFile (origen$, destino$, clave$, addrof(ret%))

Encrypts a file $ source text and as a result is generated and encrypted file $ destination. For the algorithm used $ key passed as a third argument. If ret is 0% is that something went wrong.

Requires CETCrypt.DLL file or later date 21/04/2004

w32DecryptFile call ($ source, $ destination, $ key, addrof (ret%))

Decrypt a file $ source and as a result the target file is generated $ already decrypted. For the algorithm used $ key passed as a third argument. This key must be the same as that employed in the encryption process.

If ret is 0% is that something went wrong.

Requires CETCrypt.DLL file or later date 21/04/2004

w32GetEnvironmentVariable call (variable $, addrof ($ value), addrof (return%))

Gets the value of the environment variable specified as facing. The return value is the number of characters in the value of the variable.

w32SetEnvironmentVariable call (variable $, $ value, addrof (return%))

Sets the value of the environment variable whose name is specified as the first argument.

w32ZipFile call ($ file, addrof (return%))

Compress the file and generates the. Zip file to the same directory.

w32ZipFolder call ($ folder, ignorarruta% addrof (return%))

Compress the folder and all subfolders, generating a. Zip with the same name.
Ignorarruta% parameter, if specified to 1 forces ignoraren relative paths of the files. Otherwise, specify 0

w32UnZip call ($ ficherozip, ignorarruta%, addrof (return%))

It creates a folder with the same file name and location that decompresses the content.
Ignorarruta% parameter, if specified to 1 forces ignoraren relative paths of the files. Otherwise, specify 0

w32CompileProject call ($ ruta_completa_del_fichero_prj, nombre_fichero_bat_que_se_creara_en_el_subdirectorio_bld_del_proyecto $, $ valor_de_b_cetlib, addrof (return%))

Function that generates a file. Bat in the subdirectory. Bld of a BOT project W32 specified as an argument. This file. Bat contain statements to compile the entire project.

w32CompileProject, requires Seller CETAPBLD.DLL dated 1/29/06 or later
Ejemplo:

```
call Bgetenv(addrof(lib$),"B_CETLIB")
call w32CompileProject("f:\cet-
w32\genew32\grid\grid.prj","compila.bat",lib$,addrof(retorno%))
```

call w32CreateTWAIN (padre%, id%, "VintaSoft.Twain", posX%, posY%, alto%, ancho%, addrof(TWAIN%))

Crea un control de captura de imágenes. Requiere la librería CETTwain.dll y el componente ActiveX .

Call w32PDFCreatorInit(ddestino$, fdestino$, addrof(retorno%))

Configura el spooler para que la impresión del documento que se realice a continuación resulte en que se cree un documento pdf, cuyo nombre está especificado por el parámetro fdestino$, en el directorio que se especifique como primer argumento. El valor de retorno% será –1 si se produjo algún error o 1 en caso contrario.

Requiere que el equipo tenga instalada la utilidad PDFCreator que es freeware. Requiere la librería CETActiveX.dll de fecha 22/12/2005 o posterior.

Ejemplo:

```
DEF FNConvertirAPDF$(origen$, tipo$, fdestino$, ddestino$)

call w32GetDefaultPrinter(addrof(imppre$),addrof(ret%))
call w32PDFCreatorInit(ddestino$,fdestino$,addrof(retorno%))
if retorno%=-1
   FNConvertirAPDF$="0"
   goto FINConvertiraPDF
ifend

SELECT ucase$(tipo$)
CASE "DOC"
     Rem Abrir un documento existente de nombre: origen$  (con ruta completa)
     call cetOleCreate( ADDROF(word%), "Word.Application" )
     call cetOleGetProperty( word%, "Documents", addrof(docs%) )
     call cetOleExecute( docs%, "Open",0,origen$,1)
     call cetOleGetProperty(word%,"ActiveDocument",addrof(doc%))
     call cetOleExecute(doc%,"PrintOut",0,0)
     sleep 5
     call cetOleExecute( word%, "Quit", 0, 0, 1 )
     call cetOleDestroy( word%)
CASE "EXC"
      Rem Abrir un documento existente de nombre: origen$  (con ruta completa)
      call cetOleGet( ADDROF(ex%), "Excel.Application" )
      call cetOleGetProperty( ex%, "Workbooks", addrof(books%) )
      call cetOleExecute( books%, "Open", 0,origen$,1 )

      call cetOleGetProperty(ex%,"ActiveWindow",addrof(aw%))
      call cetOleGetProperty(aw%,"SelectedSheets",addrof(ss%))
      call cetOleExecute(ss%,"PrintOut",0,0)

      sleep 5
      call cetOleExecute( ex%, "Quit", 0, 0 )
      call cetOleDestroy( ex%  )
```

```
CEND

FNConvertirAPDF$="1"
FINConvertiraPDF:
call w32SetDefaultPrinter(imppre$,addrof(ret%))
FNEND

call cForceExit
call Bgetcwd(addrof(dir$))
a$=FNConvertirAPDF$(dir$&"\cinta1.doc","DOC","Copia de cinta1",dir$)
a$=FNConvertirAPDF$(dir$&"\world.xls","EXC","Copia de World",dir$)
quit
```

RUTINA ENVIAMAIL:

The attached file contains the subroutine emailole.b ENVIAMAIL that sends an email with an unlimited number of attachments and without any intervention by the user. You will not need to be installed or configured a mail client program acting as user agent.

The performance parameters of the subroutine expects is:

$ servidorsmtp name or IP address of the SMTP server through which will be delivered to the mail. Example: "smtp.gmail.com"
puertosmtp% port number that will be shipping. This value must specify the mail service provider. Usually is 25 if not using the SSL protocol or the 465 if you use the SSL protocol.
$ E-mail to the sender of the mail. You can also specify the name of the sender. For example: "" "Sender Name" "<omegasystemstests@gmail.com>"
$ E-mail to the recipient. They can be several comma separated. You can also specify the name of the recipient with the same format as in the previous example.
Post subject matter $
$ message Message Body
$ attachments should be an array initialized with the full path of all files you want to attach.
usarautenticacion% should take the value 0 if the SMTP server does not require authentication or 1 if authentication is required, the latter value should be assigned to the following two parameters.
$ user Username with which to authenticate to the SMTP server in the event that this is required. Usually usually coincides with the email address of the sender, although the exact format you must specify the mail service provider.
$ password Password to authenticate to the SMTP server in the event that this is required.
usarssl% Indicate the value 1 if using the SSL protocol on shipping, or 0 otherwise.

Chapter 12 Development Utilities

12.1 Introduction

CET BASIC The product includes a variety of utilities that can be used in development programs and in the recovery of data files. These utilities are described in this chapter. To clarify the syntax have been used the following criteria:
[] Double brackets, an optional parameter indicator.
I A vertical bar indicates a choice between two parameters.
The italics italics indicate a variable name defined by the user.
Although Windows commands no difference between upper and lower case, all utilities except CREATE names are a mixture of both. We recommend you get used to the distinction between the two (case sensitive) especially if you go to port applications to UNIX environment, where this difference is important.
It is important to stress that these utilities are 32-bit programs to be stored in the directory W32 executable programs, by default the \ cetbin. If you have a copy of CET-BASIC product on the same system, then you will also have 16-bit versions of these utilities stored in the \ bin corresponding. Set the PATH variable to the directory selected is the first specified to avoid potential problems when trying to access the wrong versions.
There is an online help available for each utility, and such support is accessed by entering the name of the utility without any parameters. This information is stored in the file butil.msg in the same directory as the executable.
These utilities are documented in alphabetical order in this chapter.

12.2 Bcopy

Bcopy function can be used to copy or mix all records or a particular group of records from one file to another file .. The syntax of the command line is:
Bcopy [-f desdeclave] [-t hastaclave] [-c count] [-r] [-m] [-s] [-d reclen] [-i reclen keylen] <source> <target>
Where:
[-F desdeclave]
Copy starting from the specified key with hastaclave, with a key index of a file or a record of a direct file
[-T hastaclave]
Copy the source file to, but not including, the record specified by hastaclave. The default option is to stop at the end of the file.
[-C count]
Specifies the number of records copied.
[-R]
Replaces the destination file if it exists. The default option is to ask the user whether the existing file should be replaced.
[-M]
Mix the source file records within the target file. When using this option and the source file does not exist, the program will ask the user about the action to take: "Create / Quit (C / Q)", ie Create / Exit, after choosing the action press Enter to continue .
Note that this option will not change the parameters of the existing file. This feature makes it possible to copy records to a file with a password and / or a different record length
[-S]
Create a sequential output file.
[-D reclen]
Create an output file directly
[-I reclen keylen]
Create an output file indexing.
Operating Notes

1. When the-f (desdeclave) and-t (hastaclave) or-c (counter) is used in the same command line, the copy starts at the record specified by desdeclave and ends when the counter value exceeds or when reaches the specified log hastaclave as the condition is reached first.
2. When the-r (replace) or-m (mix) are not specified in the command line and the destination file exists, the program will ask the user about the action to take:

1. Replace / Merge / Quit (Replace / Mix / Out)
2. Press Desired Followed with Enter key (Press the desired key followed by Enter)

3. When you specify both-my-r option, the-m option is ignored.
4. When the destination file does not exist and none of the options-s,-d, or-i are defined, the file is created with the same type, key lengths and record the source file.
5. When using more than two sources to specify the type of source file, it will use the last option specified in the command line.
6. When you use the-m (mixed) and the key length of the source file or record is different from the destination file, use bcopy target file specifications and lengths truncate or expand properly.

12.3 Bcreate command

Bcreate utility can be used to create indexed and direct file or to delete the contents of an existing file. If you plan to create files from within programs w/32, we recommend using Bcreate function (see next chapter) instead of this command, which avoids having to leave the BASIC environment to develop this operation.
Note that this command is an enhanced version of the CREATE command, you must have used on other platforms .. If desired, you can rename the CREATE command with the name for compatibility.
Bcreate recognizes a variety of command options. The shortcuts are underlined. Note that precede signed options open parenthesis is optional.
The syntax is:
Bcreate <filename> [(] [options]
where:
filename
The file name can be either in MS-DOS and THEOS format. For example: cp \ data \ cp.data.custname custname and can be used interchangeably.
Nodesize value
Specifies the node size to use when creating an indexed file. The default is 512. Although w/32 programs can read files with the default size of the node, we recommend that you use the "n 1024" for consistency with Bcreate function.
indexed
Creates an indexed file which is actually two files: a. Idx for keys and file records. Dat for data.
direct
Create a direct file (on access)
reclen
Assigns a record of length n.
keylen n
Assigns a key of length n. This option is ignored for direct files.
clear
Clears an existing file content. The key and record length will remain the same unless you specify other options keylen, reclen and / or nodesize in the command line.
Note that the file must be closed prior to this to work successfully.

All files are dynamically assigned BASIC CET so that the option of THEOS file size is ignored. If the command is used to create key files THEOS program then creates an indexed file.
The following examples will create files indexed data \ custname.idx and data \ custname.dat with a key length of 11 and a record length of 350 bytes:
Bcreate data \ custname indexed reclen keylen 11 350
Bcreate data \ custname.data indexed reclen keylen 11 350

12.4 El comando Blist

Blist command can be used to display the contents of a file indexing, direct or sequential. A message will display the name and type of the file.
The syntax of the command below. Note that by entering the command without parameters displays a command line help.
Blist [-h] [-f desdeclave] [-t hastaclave] [-c count] [-l length] [-p page] [-m margin] [-field numbers] [-v] [-u] <filename >
<file_name>
Specifies the DOS format file or THEOS. For example, any of the following apply to list names cp \ data \ name:
. \ Cp \ data \ name (or \ cp \ data \ name) cp.data.nombre
[-H]

Displays the contents of the file in hexadecimal format.
[-F desdeclave]
Start a list from the registration key to the registry desdeclave hastaclave keyed, with a key index of a file or a record of a direct file. The default is to start at the beginning of the file.
[-T hastaclave]
Finish the list in the registry (not including) hastaclave key. The default option is to terminate at the end of the file.
[-C count]
Specifies the number of records copied.
[-L length]
Specifies the maximum length of the print line. The leftover characters are printed on the next line. The default length is 79.
[-P page]
Specifies the length of the output page. The default length is 24 rows. Use "-p 0" to disable paging.
[-M margin]
Specifies the number of characters to indent the second and subsequent lines of a multiline record. The default is 5 characters.
[Numbers-field]
Specifies the field numbers to be listed in a formatted file. The default is to list all the fields in the record.
The first data field starts with the number 1. The keys of an indexed file and the registration numbers of a direct file is always displayed.
[-V]
Enables verbose mode (expanded) to show in the listing the number, type and length field before the contents of each field. The field type is for a string S, F for Float, or Integer Ito. For example:
123-45-6789 (1S17) DGRamírez, Inc., (2H11) Don Gillet (3s17) ...
[-U]
Specifies that the files are formatted UX-BASIC.
The content of an indexed file is displayed in alphabetical order key. Direct files are displayed in ascending numerical order.

12.5 Bpretty command

Bpretty utility can be used to rearrange the spacing and hierarchy (or indentation) of sources CET BASIC programs to reflect its structure.
Note that the syntax and operation of the 32-bit version of Bpretty is unlike those available in other products of CET-BASIC. The entry will not be accepted from rear. The source file name must be specified on the command line. The specified file will be rewritten, after storing the original code in a file extension. Bak. By default, the backup file has the same name as the source file. (file names must be in DOS format). The syntax is:
Bpretty-n-p-i fuente.b [fuente.bak]
where:
-n Indents n spaces. The default is 4.
-t Replaces spaces with TAB if possible.
-I Insert the include files before performing any other operation.

12.6 Brenum command

Brenum can be used to renumber or remove the numbering CET BASIC source programs up to 999,999 lines. The file names must be in DOS format. The syntax is:

Brenum [-a] [-n initial] [-i | + increment] [-b begin] [-e end] [-u] <fuente.b> [fuente.bak]

where:

-A

Makes all lines are numbered range. When omitted, the default numbered lines only with numbers.

Note that the lines of code contained in include files are not renumbered, even when using this option-a.

-N

Start renumber from given initial number. By default this is 10.

-I

Increases line renumbering by the increased value given in the command line. The default is 10.

+

Works like the-i option to specify how the numbers will increase.

-B

Renumbering starts from given initial line. When this is omitted, the default is the first line of the program.

-E

Ends at the finish line Renumbering given on the command line. When omitted, the default is the last line of the program.

-U

Removes all line numbers from the source file.

Renumber Note that portions of a file can cause, that these lines are moved to preserve the previous numerical order.

The options can be in any order given that the parameters associated with these options should be in correct order. For example, the following command lines renumbered all fuente.by file the result of its output goes to file dest.bak

Brenum-to fuente.b des.bak

The following examples all produce the same results: fuente.b be renumbered from line 200 to line 300. The new lines renumbered begin with the number 600 and will increase to 5 on 5. The result goes to xfuente.bak

Fuente.b Brenum xfuente.bak-NIBE 600 5200300

Brenum fuente.b-b 200-e 300-n 600-i 5 xfuente.bak

Brenum-n600-be +5 200 300 fuente.b xfuente.bak

12.7 CETLNK32 command

CETLNK32 the program used to create a library of object files that can linkarse the main program using the library flag. The syntax is:
cetlnk32 / lib / out: libreria.lib fich1.obj ... | *. Obj
library:
Specifies the file name bookstore to create
file1
Specifies the name of the object file to include in the library
*
Indicates that all object files the current working directory must be included in the library
Both examples miobjlib.lib create a library called the current working file:
cetlnk32 / lib / out: miobj.lib dg3btnmsg.obj dgerror.obj dgkey.obj
cetlnk32 / lib / out: miobjlib.lib *. obj
After creating the library, enter in the autoexec.bat PATH LIB directory. You can then compiling your program linkarlo main library with the flag in this way:
Mainpgm.b mainpgm OBW32-o-lmiobjlib
CAUTION: Due to the problem of the Microsoft LIB utility, you must delete the current library before reinvoke CETLNK32 command to build the library and include a new object.

12.8 Dcheck command

The product uses CET w/32 D-ISAM, an indexed file manager of Byte Designs. D_ISAM format files are compatible with files C-ISAM format, the file structure that is used in other products CET BASIC. (Currently, there are no available version of Windows Product Informix C-ISAM).

Dcheck is a D-ISAM utility designed to test and adjust inconsistencies of an indexed file. Enter the command without arguments to see the usage syntax. It is important to note the following:
• Dcheck only accept filenames DOS.
• Dcheck can be used to check the structure of files, list the contents of the keys and the order of the data file and checking the consistency of the file, without modifying it.
• Dcheck can build an index structure and a data file. Delete records that have not been cleared physically within the file and still held within the same site.
The syntax is:
Dcheck [-] [i] [l] [n] [y] [q] [b] [h] [x] [o] <fichero_isam>
where:
[I]
Check the index only. There is no checking of the data.
[L]
List the contents of the indexes on screen.
[N]
Do not try to rebuild the index.
[And]
Try to rebuild the index if it is imperfect.
[Q]
It runs in 'silent' messages showing exclusively necessary.
[B]
Rebuild the index from the data, even when the index is well. To build an index for data from another source:
1. Organize the data in a file with each record separated by a newline character (ASCII 10)
2. Creates an indexed file with a different name, with the desired key. No need to add the data.
3. Move your data file to the file. Dat for indexed file created again with:
dcheck-b isamfichero
This will make your data file when building is fully indexed ISAM file
[H]
Only print the file structure without check it. This option does not require exclusive access to the file.
[X]
Displays the contents of each record as hexadecimal.
[Or]
Prints a list of numbers that refer to the registration number to appear with indices. This feature can be used with the-q option to create a print filter quick.
If you find an error, or if you use the-b option, removes Dcheck indices, and reconstructed from the data. The data is never destroyed in the process of reconstruction.

12.9 BSORT command

Bsort classification is a very flexible tool designed to process selected items CET BASIC data files and list them in an orderly sequence. It is available for use with all products CET.

Bsort utility management is a record-level full of options. Allows a file sort by any field, subfield, or combination of fields. The input file organization can be sequential, direct or indexed and can contain both binary and ASCII text records. The output file can also be sequential, direct and indexed.

Bsort is fast and efficient and can be used to enhance any application CET. For example, business programs written in BASIC CET users can sort files and return to BASIC for the show.
Some of the features are:

- Bsort command parameters can be stored on disk. This feature allows to change parameters without recompiling the BASIC program.

- Bsort can "jump" to the first 255 records from the input file. This is especially important when you do not want to include control records or headers with specific program information in the orderly exit.

- The specification of the position of the character field or subfield allows to classify the fixed fields of variable length. The quick key classification is implemented.

- Outputs can write complete records, the key field or any portion of the record, according to programming needs.

- You can write out any combination of fields (or subfields) regardless of the order in which the records are classified.

- You can override specific fields within the output record.

- There is the possibility to use up to 32 instructions to select specific records that meet the desired criteria. So just be sorted and written to the output (as specified) these records previously selected.

- In direct or indexed files, you can select a range of records based on the key value.

- You can specify alternate directories for temporary work files.

12.9.1 Instalation:

Bsort income is distributed as an executable file with an error message. The Bsort executable in a directory search path. Typically, this will be the same directory you use to store other programs CET.

The error messages caused by Bsort are stored in / lib / Bsort_err (UNIX) or \ lib \ bsort.err (DOS). While the operation does not require file Bsort error, this should be located in the / lib whether to display error messages.

12.9.2 BSORT sintax

Bsort parameters can be indicated from the command line, from a judgment CSI / CSH within a CET BASIC program or from a text file sequentially. The command syntax is:

Bsort infile outfile [dirtrabajo {especlave [espesel [espesal}

Where:

infile

is the name of the file containing the data to be sorted. The filenames can be in the syntax of the operating system in use or in THEOS format. (No names are supported library files THEOS)

Bsort also allows classification parameters stored in a sequential ASCII file with extension. Qs. When this type is specified in the input file, the remaining line arguments are ignored.

Bsort seek the input file in the directory specified by the environment variable B_FPATH, and then in the current working directory in the same way that files are searched in the CET BASIC programs.

See the next section for details on the types of input and output files that can be used.

outfile

is the name of the file where to store the sorted data. By default, the output file will be a direct file except when the input file is sequential and not use the limited output option. In that case the file will also be sequentially output.
You can change this default behavior when using output option to specify the file type, either sequential, indexed, or direct the same type as the input file.

dirtrabajo

is an optional parameter that can be entered to indicate the directory to use if work files are stored on disk.

Note that by default requires Bsort \ tmp directory on the current drive. The TEMP environment variable can also be used for other pathname.

especlave

defines the type of the input file and the fields to be sorted. You can specify up to 32 fields. The order establishing the definitions of the fields is the same by which be sorted later. The specifications of the sort keys are entered in the following format:

{[B [# nnn, [field # desc-field ... }

Where:

{}
indicates the beginning and end of instructions. Under UNIX, the opening brace may have a special meaning to the shell. Then type a backslash (\) before the character.

B
used at the beginning of the especlave to indicate that the input file is formatted binary records. For clarity you can use a space between this and the next option (eg # 80). For example:
Bsort infile outfile {B # 80, ...}

Nnn
is the maximum record length that must be specified when the input file is sequential.

field #
is the position of the field within a record containing binary data format to be used in the sorting operation. The key field in indexed files are preceded by a 0, and the data fields with 1. Note that this parameter should not be used for ASCII files as these are composed of raw logs (eg a string field per record).

Off-field
descriptor is used to describe the contents of the field. You can specify the following:

F indicates a floating point number. (This parameter is used only in binary logs. ASCII files consist of data strings.)

I indicates that the data is an integer value. (This option is used only in binary logs.) For example, to sort the input file for the field # 3 containing an integer value expressions are valid:

{B3I ... or {B 3I.

-From-to [N indicates a string that begins with the character position start and ends with the last character position of the field for which you want to sort. To sort by the string consisting of the first 5 characters that compose the field 6 enter:

{B6-1-5, ... or {B 6-1-5

N The option may be used to indicate that the string is a number which must be converted to determine the grade range correctly.

If the chain length is less current than specified, data will be added trailing spaces.
The total length is specified can not be greater than the key field when the key is an index file, or greater than the length of the record when it comes to data fields.

Note that before classifying indexed files ASCII format, the registry key is concatenated to the data field. This means that any specification from-to be displaced by the key length. It also means that the key can develop in all normal sorting options.

D indicates that the field be sorted in descending order instead of ascending order is the default.

A file can sort by up to 32 fields using a single command. If more than one field, separate each with a comma especlave instructions or space. Examples are included in the next section and appendix.

espesel

The specifications of selection can be used to select records from the input file by the specification criteria field. Just be sorted and written to the output file records that meet the criteria.

Enter the specifications in the following format using commas or spaces as separators. The field # (in binary format files) and off-field variables used in the same manner as previously noted.

S [field # desc-field [operator, 'constant', ... , [OR, ...

S
specifies that the following instructions are to be used to select the records to be sorted. This character should precede immediately (without comma and space) the first definition of field selection. This instruction terminating when entering an instruction L, P or Q or when it reaches the end of the command line.

operator

indicates the relationship field data must comply with the 'constant' for the record to be selected.

EQ - equal to
NE - not equal to
GT - greater than
GE - greater than or equal to
LT - less than
LE - less than or equal

'Constant'
defines the constant to be used in the comparison. Since the constant must be enclosed in single quotes, it can not contain this character. Under UNIX, the single quote has special meaning to the shell and must be preceded by a backslash character (\).

If the data field being compared with the constant is defined as numeric, then the constant Bsort convert a numeric value.

OR
is used to indicate the beginning of a selection criteria a set of alternate registers. If omitted, when performing multiple specifications shall be selected that are linked by the AND link.

espesal
Output specifications that can allow up to 32 fields indicate the check is written to the output register. These fields are completely separate from those used for classification. If you omit this option, the full log entry is written to the output file. For this reason, this option is also called limited output option.

Enter the specifications in the following format. Note that the field variables # and off-field are used in the same manner as mentioned under section especlave except that no longer needs the option to N strings.

L field # desc-field ...

Where:

L
determines that the instructions specified below indicate which fields are written to the output file. The character should immediately precede the first definition of the output field with or without a space separator. This option specifies ends when an instruction S, P or Q or reaches the end of the command line.

Pnnn

indicates that the output file will not include the first records nnn (1-255) of the input file. This option should not be used when working with files indexed since Bsort not read the records in the sequential order of the file key.

Q
indicates Bsort to run in silent mode. When using this option, not displayed or the copyright notice or the number of records.

O
indicates the access mode to be used in the output file. The syntax is:

L field # des-field ... ... O = type

The valid encodings file types are:

R - the same type as the input file
S - sequential file
D - Direct File
I - Indexed file

Note that when this option is not specified, the previous rule will be used for the output files.

C
is the continuation character indicating that the command description continue on the next line. This option is used by introducing C after the comma that ends the previous instruction and then <retorno of carro>.

Note that this option can only be used from the operating system prompt or from a file. Qs.

The classification process.

There are three main stages in the management process. The first phase is the specification phase, in which Bsort get the information needed to carry out the task.

In the second phase, Bsort reads the input file, loaded into memory selected records and sorts. If the total size of records is greater than the available memory is created and used disk work files.

In the third phase or phase-out, records classified and / or (sub) fields specified are written to the output file indicated in the instructions for ordering.

The specification phase

During the specification phase, Bsort collects operation parameters needed to perform the classification task. These parameters can be obtained from the command line, from a judgment CSI / CSH in BASIC or a sequential ASCII file. When ordering parameters occupy more than one line of the console, use the following option (C) to inform Bsort that will introduce more input.

The classification parameters include an input file, an output file (optional), key specifications, or any selection or output options discussed in the previous section. (The specification and using the classification parameters are described in detail below)

Bsort then analyze the command line to determine if you have entered all the required parameters, and the syntax is correct. It locates the input file and checks whether the correct format.

Input file

You must specify the name of the input file. Normally file names must have the syntax of the operating system in use. Names are also permitted as cust.data THEOS format. In this case the name will become (eg / DATA / CUST)

Bsort seek the input file in the directory specified by the environment variable B_FPATH, and then in the current working directory in the same way files are searched in the CET BASIC programs.

The input file can be any standard sequential CET direct or indexed sequential (ISAM). Data records can be formatted or not, but all records of the file should be similar. Pnn Note that the option can be used when necessary ignore control registers that are at the beginning of the file.

Bsort also allows you to store the sort parameters in a special ASCII sequential file with a. Qs. When you specify this type of file as input, ignoring the rest of line arguments.

Binary format files

A formatted file containing records with alphanumeric fields, integer or floating point specified in a special data format. Each field starts with a format code indicating its type. Records with binary format are supported automatically by CET BASIC if written using the WRITE statement.
CET Blist command displays log data fields separated by commas.

Plain ASCII files

Plain ASCII files contain records made up of ASCII characters only, no field delimiters. The entire record is written as a single string. The end of the record delimiter is a carriage <retorno> or <linefeed> (DOS) or character <linefeed> under UNIX.

ASCII files are usually created with the CET BASIC PRINT statement. Each record consists of 'fields' adopting a fixed length so that the field start positions remain constant throughout the file. Often trailing spaces are added to the fields in this type of record to maintain their position in it.

Output Files

By default, dump Bsort seeks records sorted in an existing direct file. The output is directed to a sequential file when: 1) the existing output file is a sequential file exists, or 2) the input file is sequential, and not use the limited output option (L).

You can use the output option to change this behavior and to specify the file type to be used. In that case the output file can be sequential, direct, indexed, or the same type as the input file.

Bsort detect an error if there is no output file having used the exit option in the specifications.

Specifications of the sort key

They should put an opening brace. These instructions define the type of input field and the fields to sort.

When Bsort will order a binary file with the first option should be the letter "B". You can use a space to separate the character of the next option, but do not allow a comma. For example:

{B 5 ... Field # 5 in a binary format file or direct indexing
{B # 500, ... a sequential file in binary format with a maximum record length of 500 bytes.
{B 01/03/10 ... the first ten characters of the string in the third field of a record in binary format.

If the input is sequential file must specify the maximum length of the record to be read. For example, B # 500 indicate a sequential binary file with a maximum record length of 500 bytes. If detected Bsort record with a length greater than the specified error is generated.
It should also be specified sort key to set the / the character / s, data field or subfield in the record of the input file on which will actually perform the sort. It allows up to 32 sort keys. When more than one key the first is the most significant and determine the sequence of the output file.

The key specifications are determined both by the fact that the input file is binary type like plain ASCII contains records.

The records are binary field delimiters to separate each field. Records can contain strings, whole fields of floating point numbers. To order a binary file, you must indicate it location of each field containing data that will be used. If the data is of type integer or a floating point value, you must also specify a field descriptor as shown in the following examples:

2F: a floating point value found in the field # 2
3I: an integer value found in the field # 3
6: all characters in a string field # 6
6-1-5: the first 5 characters of the string field # 6
6-1-5D: the first 5 characters of the string field # 6 sorted in ASCII descending order.
6-1-5N: the first 5 characters of the string field # 6 converted to its numerical value.
6-1-5ND: the first 5 characters of the string field # 6 converted to their numeric value and sorted in descending order.

Note that you do not need any special character to indicate fields of type string. Note also that part of the chain can be used for sorting purposes. If actual data are lower than those specified are added to the right spaces. Specifications are not indicate lengths greater than the actual length of the key when working with indexed file keys or total record length when working with data fields.

The "N" is used if the string contains a valid number, the digits 0-9, are preceded by a minus or plus sign or take a decimal point. Bsort convert the string to a number for sorting. If they would do so management would be done using the ASCII value of the character string.

By default, in ascending order Bsort. If you specify the "D", the output will be sorted in descending order with the highest value first.

The sort of plain ASCII files is much simpler. If the input file is sequential, the maximum length of the record to be read must be specified as mentioned above. For example, # 500 indicates a plain ASCII sequential file that has a maximum record length of 500 bytes.

Since each record consists of a single string field, fewer options to define the specifications of the sort key. You can only apply the options from-to, N and D. For example:

1-5: the first five characters of the registration
1-5D: the first five characters of ASCII record sorted in descending order.
1-5N: the first five characters of the record converted to its numerical value
1-5ND: the first five characters of the record converted to their numeric value and sorted in descending order.

Note that there is a special case when ordering an indexed file. The registry key is concatenated to the data field before the records are sorted. This implies that any specification-up from the field indicating a data portion to be displaced by the key length. This likewise implies that all options can be made normal on key.

Selection Specifications

Bsort allows you to select the records in the input file based on the contents of the fields. Only records will be sorted to fit the specified criteria.

The specifications of the selection criteria begin with the letter "S" followed immediately by the numeric field (for binary files) and the field descriptor as explained in the previous section.

Operator is used to indicate the relationship must have the data in the field on a 'constant' before being selected.

Valid operators are:

EQ - equal to
NE - other than
GT - greater than
GE - greater than or equal to
LT - less than
LE - less than or equal to

The constants used in the comparison must be quoted, so a quote can not be part of the element to compare. Under UNIX, the quotes have special meaning to the shell and therefore need to be preceded by a slash backslash \.

If the data field to compare with the constant is defined as numeric, then the constant is converted to digital format for comparison.

The OR keyword is used to indicate the beginning of a record selection criteria alternative. If omitted default AND applies when using multiple selection criteria.

The following specifications are used to select records by way of example:

S4-1-1, EQ \ 'A \': the first character of the field # 4 is an 'A'.
S6, EQ, \ "CA \ ': the data in field # 6 is equal to' CA '
S9-1-3N, LT, \ '100 \ ': the value in the first three characters of the field # 9 is less than 100.

Although the file format may be necessary to indicate that the first records nnn (1-255) in the input file will be ignored. This feature is especially important when the header records are used to control the operations of the BASIC program.

Note that Bsort not read an indexed file in key order, so this option should not be used with this type of files.

The management phase

After they are established and verified the sort parameters Bsort reads the input file and select records according to the specifications, if specified record selection specifications.

The records are then loaded into the memory available to your organization. If the total number of records takes up more space than available memory creates a temporary file on disk. If the option is specified working directory to the working file will be created in that directory, if not specified uses the directory in which the file is input.

The output stage

In the third stage the sorted records are written to the output file specified on the command line.

Output specifications are used to limit the fields from the check that will be written to the output file. These fields are completely independent of those used in the organization. If this option is omitted the check will be written entirely in the output file.

The letter "L" is used to indicate that the following statements define which fields are displayed. The character should immediately precede the first definition of the output field without separation of space or comma. This option ends when you enter a command S, P or Q or it reaches the end of the command line.

The field variables # (for formatted binary records) and off-field have the same use as before, except that at this stage no longer needed the option N to convert a string to a numeric value.

You can also use certain options (P, Q and C) to modify the behavior of the utility Bsort and indicate that you need to ignore records starting (control) of a file, not to display the copyright information and the number or records management instructions have been introduced into more than one line.

Bsort examples

This section illustrates some typical commands for sorting records in a UNIX environment. The equivalent commands in MS-DOS could use a backslash (\) at the start of the file names and would not require a backslash single quote to delimit a selection specification.

Binary Sorting Records

Example 1:

In the first example,. / Testfile is a sequential file created with the WRITE statement. Each record contains six fields. The first field of the record is of type integer and the rest of the fields contain strings. The maximum record length is 80 bytes.

Bsort. / Testfile. / {B # 80,6-1-1 out, ,2-1I 1-5}

This command instructs testfile records and dumps the entire record a new file called out as sequential not used the limited output L option.

First, the records are sorted by the first character in the sixth field in ascending order. The records are then sorted by the integer value of field 1. Management is performed using third the first five characters of the string that is stored in the second field.

Example 2:

The following command is similar to the previous one except that the records are sorted by the first five characters of the second field after previously been turned into numbers.

Bsort. / Testfile. / Out {B # 80.1 I-1 ,2-5N}

Example 3:

This example selects records. / Testfile having an A as the first character of the field 4. The selected records are then sorted by the first 40 characters of field 4. Only these characters are written directly to the file named. / Out.

Bsort. / Testfile. / Out {B # 80 01/04/40 S 4-1-1 EQ \ 'A \' L 01/04/40}

Note that Bsort not written to a sequential output file if specified limited output option L.

Example 4:

This example shows how to specify that the sorting parameters in a sequential ASCII file. This feature is especially useful when you want to change the sorting parameters without recompiling the BASIC program. Also enter long classification instructions that require the use of the continuation option C (to prevent exceeding the maximum limit of 127 characters in a command that requires MS-DOS).

Should consider two requirements to use this technique. First, the file name must be a name with UNIX or DOS format, depending on the operating system in use. The file name must have the extension. Qs. For example:

Bsort. / Example.qsp

The contents of two records in a file. Qsp might be:

. / Testfile. / Out {B # 80,6-1-1,3-1-6,4-1-40,5-1-4,1 I, C
2-1-5, S4-1-1, EQ, \ "A \, L4-1-40, 1I}.

Example 5:

In this example, the file to be sorted is called. / Isamfile. Is an indexed file with three fields in each record. The first field contains an integer, the second a floating point number and the third a string.

The following command will order the contents of. / Isamfile and write the sorted records in a file called direct. / Out in descending order. The records sorted first by the first five characters of the third field. The second and third sort criteria will be determined by the contents of the second and third fields.

Bsort. / Isamfile. / Out {B} 3-1-5 2F 1 ID

Sorting Records ASCII

This section illustrates some typical commands for sorting records in ASCII UNIX environment. The equivalent command in a DOS environment could use a \ (backslash) at the start of the file names and not required to enclose a single quote character in a selection specification.

Example 1:

In this example, the file is sorted,. / Testfile, is a sequential file and each record contains a single string of 40 characters in length.

Bsort. / Testfile. / Out {# 40, 1-4}
The records would be sorted by the first four characters. Each full registration. / Testfile be written to a sequential file called. / Out.

Example 2:

The following command isamfile order records, one indexed file key with a length of 4 and a record length of 40. Since this is an indexed file, before the operations take classification is concatenated to the registry key data fields.

Bsort. / Isamfile. / Out {5-9} L1-4

Bsort sort the records in the input file using the first four characters of the data field and write the registry key to a file named index. / Out.
APPENDIX

Error Messages

The Bsort error messages are recorded in / lib / Bsort_err (UNIX) or \ lib \ bsort.err (DOS). While for the operation is not required Bsort file error, this should be in the directory / lib whether to display error messages.

The following is a listing of the contents of the error file:

```
1. Insufficient memory.
2. Unable to open file "{0}".
4. Can not create file "{0}".
5. Option invalid: "{0}".
6. Unable to locate the file "{0}".
7. Unable to unlink file "{0}".
11. Reading Error: {0}.
12. Writing Error: {0}.
54. Syntax error.
```

The following errors are specific Bsort.

```
501. Unable to close the file "{0}".
502. Failed conversion STRING to REAL.
503. Data type not supported or invalid: {0}.
504. Invalid data type, INTEGER expected.
505. Invalid data type, REAL expected.
506. Invalid data type is expected STRING.
507. Search Error (L): "{0}".
508. Sequential record exceeds the buffer size
509. Output File not found.
510. Option separator is required.
511. Avoid a meter higher than the record in the file counter.
512. Key field in a non-ISAM file.
513. Option 'N' employed in a field that is not chain.
514. Relational operator invalid: "{0}".
515. No comparison is constant.
516. {0}: substring record size exceeds the file: {0}.
517. {0}: substring key field length exceeds {0}.
518. {0}: field FROM> TO field.
519. Records ignored = {0}.
520. Records read = {0}.
521. Selected records = {0}.
522. Written records = {0}.
523. Program interrupted (signal = {0}).
```

Chapter 13 Routines performed in C

13.1 Introduction.

The Application Development System w/32 CET has a wide variety of C language routines that allows to perform operations that would be impossible or very difficult to program them in BASIC only.
When using functions note that:
Differentiate between upper and lower case is important. The function name should be written as it appears in the manual.
The file names must appear in DOS format or format THEOS. CET is a name with a period as a format and converts relevant. For, example, the following names are considered identical.
cp.data.nomfich cp \ data \ nomfich
If the DOS file included in point, using the following format:
. \ trans.fil
This chapter describes some of these routines. See Libraries CET BASIC Manual for more information on available features.

13.2 Routines performed in C

The routines currently supported are documented in this chapter. As developers will need additional functions, they will provide new routines. Contact your dealer CET, if you do not find any of these routines.
For your convenience, the routines have been classified by their functionality.

13.2.1 Functions of files

The following functions are provided to avoid having to use operations CSI / CSH. Any error that generates an error ERROR ON catchable by using the list of errors found in the file brun.err.

13.2.2 Bcreate

Bcreate utility operates as described in the previous chapter (Bcreate) except running within a BASIC program using a CALL statement instead of a CSI. The syntax of this statement is:
CALL Bcreate ("create command line")
Keep in mind the distinction between uppercase and lowercase letters, in the same command syntax. The function name must be entered exactly as shown, taking into account the uppercase and lowercase. The name of the file to create or delete can be in any format DOS or THEOS. For example:

CALL Bcreate ("data \ nomfi indexed keylen reclen 350 11")
CALL Bcreate ("indexed reclen nomfi.datos keylen 350 11")

Both commands created data \ nomfi indexed as a file with a key length of 11 and a record length of 350.
The following example illustrates how to use Bcreate to clear the contents of a file. Note that the file must be closed before deleting its contents.

```
CALL Bcreate ("cp \ data \ nomfi clear")
```

13.2.3 Berase

Berase is designed to delete a file without having to execute the judgment CSI with the DOS command DEL. In any case, the file must be closed before it can be deleted. The statement syntax is:

```
CALL Berase ("filename")
```

The file name can be formatted DOS or THEOS. If the file is indexed both. Idx as the. Dat will be deleted.

13.2.4 Brename

Brename function is designed to eliminate the execution of the judgment CSI with the DOS command RENAME. In any case, the file must be closed before being renamed. The statement syntax is:

```
CALL Brename ("fichero_original fichero_nuevo")
```

The file name can be formatted DOS or THEOS. If the file is indexed both. Idx and file. Associated dat be renamed at once.

13.2.5 Bcopy

Bcopy function can be used to copy a file without having to resort to a judgment call CSI DOS COPY command through a CSI. The names of the files can be formatted in DOS or THEOS format. If the target file does not exist, it is created with the same specifications as the source file.
The CALL statement syntax is:

```
CALL bcopy ("Fichero_fuente Fichero_destino")
```

Note that this function does not have all the features offered by the command of the same name (see the previous chapter).

13.2.6 Bfilestat

This function is used to determine the existence of a file or directory. The syntax is:

```
CALL Bfilestat ("filename, ADDROF ($ state)")
```

The parameters are defined as:

filename The file name must be in DOS or THEOS. If the file is indexed, the extension (. Idx or. DAT) should be specified, eg \ data \ test.idx.

It's been a string returns a directory D, R to a file or null if the filename does not exist.

Note that Bfilestat function has been available in the past as filestatus. The latter function is also supported but only recognizes DOS file format.

13.2.7 Bgetcwd y Bchdir

Bgetcwd function is used to get the current working directory (CWD). Bchdir is used to change the CWD (current working directory). It is important to note that this change is permanent. The statement syntax is:

```
CALL Bgetcwd (ADDROF ($ cwd))
CALL Bchdir ($ cwd)
```

13.2.8 Bindinfo

This function can be used to obtain information about open files indexed. The syntax is:

```
CALL Bindinfo (channel% addrof (counter), addrot (reg%), addrof (key%))
```

where

channel Is an integer variable or constant that represents the channel at which to open the file.

counter It is a numeric value that returns the number of records in the file.

Reg The data record length. Note that this value has traditionally been the addition of more data characters of the key

Key This value returns the length of the key.

13.2.9 Functions related to the environment

13.2.10 Bgetenv y Bputenv

The function can be used to obtain Bgetenv charges any environment variable autoexec.bat file. The statement syntax is:

Bputenv can be used to activate the environment variables to a value only during program execution. (Check with your dealer about which variables that may change). After the program ends, restores the original value.

CALL Bgetenv (ADDROF ($ value), $ var)

CALL Bputenv (comandoset $)

The parameters are:

$ value Returns the information stored in the environment variable or a null character if the required information is not available. (For example if this variable was not updated with any value).

var $ Specifies the name of the environment variable

$ comandoset Specifies the command to update the environment variable with a new value.

The following code shows the use of these features:

```
CALL Bgetenv (addrof ($ s), $ aa)
PRINT "Present value" aa $; "is <" $ s ">"
REM "Now change the environment variable in the form <VAR> = <LIST>"
$ new = "PATH = C: \ DOS; \ C: \ BIN"
CALL Bputenv (new $)
```

13.2.11 Bgetenv32 y Bputenv32

Bgetenv function can be used to obtain or upload any file environment variable W32APBLD.w32 The statement syntax is:

Bputenv32 can be used to activate the environment variables to a value only during program execution. The parameters and syntax of these functions are similar to functions and Bgetenv Bputenv.

13.2.12 Bmemavl y Bmemmax

These functions are supported only to provide compatibility with other products CET.

13.2.13 Bsernum

Bsernum is used to obtain the serial number of the protection key (dongle) CET. See SERIAL OPTION statement in the Language Reference Manual CET BASIC for another method of protection.
The statement syntax is:

```
CALL Bsernum (ADDROF (serno%))
```

The parameters are:
serno% Is the serial number programmed into the dongle CET.

13.2.14 Functions of Program

13.2.15 Becho y Bnoecho

Bnoecho Becho and can be used to enable or disable the character echo during operations LINPUT or INPUT. No arguments used. The syntax of this statement is:

```
  CALL Becho
  CALL Bnoecho
```

13.2.16 Bkdshift

Bkdshift function can be used to check the status of shift keys or switching within the PC keyboard. These include the Shift Right, Left Shift, CTRL, Num Lock, Caps Lock, and Insert. Note that the sequence ALT / <key>, not supported because they are special key combinations used by Windows. The statement syntax is:
CALL Bkbshift (addrof (key%))
The parameters are:
key Returns an integer encoded where each bit represents the status of the keys as:

Bit Activo	Condición
0	Está siendo pulsada mayúsculas Derecha
1	Está siendo pulsada mayúsculas Izquierda
2	La Tecla CTRL está activa.
3	(no soportada)
4	Bloq Despl se ha activado.
5	Bloq Num se ha activado.
6	Bloq Mayús se ha activado.
7	Insert se ha activado.

Por ejemplo:

```
PRINT crt ("clear")  \  PRINT "Pulse Shift, Ctl, ALT o otra tecla [y CR]:"
PRINT "Drcha    Izq                      Bloq   Bloq   Bloq          "
PRINT "Activa   Activa    CTRL    ALT    Despl  Num    Mayus  Insert "
PRINT "_____    _____     _____   _____  _____  _____  _____  ______ "
WHILE 1
       CALL Bkbshift (addrof (key%))
       if key% <> 0
              Rshift% = key% and 01H
              Lshift% = key% and 02H
              Ctrl%          = key% and 04H
              Alt%           = key% and 08H
              ScrollLock%    = key% and 10H
              NumLock%       = key% and 20H
              CapsLock%      = key% and 40H
              InsMode%       = key% and 80H
              PRINT at (1,7);
              IF (Rshift%           ) then PRINT tab ( 3);crt ("rvon")+"
       "+crt ("rvoff");
              IF (LshIFt%           ) then PRINT tab (10);crt ("rvon")+"         "+crt
("rvoff");
              IF (Ctrl%             ) then PRINT tab (17);crt ("rvon")+"         "+crt
("rvoff");
              IF (Alt%              ) then PRINT tab (24);crt ("rvon")+"         "+crt
("rvoff");
              IF (ScrollLock%       ) then PRINT tab (31);crt ("rvon")+"         "+crt
("rvoff");
              IF (NumLock%   ) then PRINT tab (38);crt ("rvon")+"         "+crt ("rvoff");
              IF (CapsLock%  ) then PRINT tab (45);crt ("rvon")+"         "+crt ("rvoff");
              IF (InsMode%   ) then PRINT tab (52);crt ("rvon")+"         "+crt ("rvoff");
       IFEND
       GET a%
       IF a% <> 0 THEN BREAK
WEND
```

This chapter has covered some of the C functions that can be called within BASIC programs. See Libraries CET BASIC Manual for information on other features available.

Chapter 14 Implementation Notes.

14.1 Introduction

This section has been created to cover the generic information you need to deploy a Windows "w/32". The application window size and location. When running a program w/32, this entire window used by default. The starting position is x = 0 and y = 0 (in coordinates relative to points). You can change this position by passing the argument-wposX.Y within line arguments when you run the program. For example, the following command opens a window starting at position 100.50 (points):
test-wpos100, 50
The initial size can be changed using the command line option-vsizeX, Y.
To facilitate the concept of `points' (pixels), remember that a character displayed in the default font courier 9-points, is about 8 points wide and 13 points high.

14.2 El Default PIF file

Each time you run a DOS command from within Windows, by default the command will run DOS full screen mode (although you can always press ALT-ENTER). This also applies to programs w/32 running MS-DOS commands through CSI / CSH.
When you call DOS normally tends to produce a "flash" because the screen is cleaned to remove the program in a DOS window. When the program ends, you will see another "flash" due to the reverse process.
To avoid this "flash", change the default Windows to run DOS programs in a "hidden", instead of using the entire screen. This is done using the PIF editor to modify the file default.pif. If you change the property "Display Usage" to "Full Screen" to "Windowed", each time you run a DOS program, a small window that puts DOS. When the program ends, this small window disappear.
If you have a specific program you want to run DOS in "full screen" from within Windows, we suggest you create an icon for the program and use a. Pif, which specify "Full Screen" for screen printing. For more information on the program icons and files. Pif, refer to your Microsoft Windows documentation.

14.3 Current working directory

Any file (data) to open a program w/32 must have specified an absolute path or relative to the current working directory (CWD current working directory).
The current working directory (CWD) can be specified by two roads. When a program is executed from the File Manager, the CWD is the directory where the program began to run. If the program is executed from the Program Manager (after adding the group
programs) then the CWD is part of the Program Manager Properties. You can change or view the properties by selecting the icon from the File menu.
Included are two routines done in C, and Bchdir Bgetcwd to determine and change the CWD CWD. See chapter C language routines performed for specific information.

14.4 The name of a file

The file names can be specified in DOS or THEOS format. In either of the two formats must maintain proper nomenclature filename.

A DOS format is recognized when the prefix "\", ". \", Or the unit code predating the filename. The file path can be fully qualified as \ pos \ cp \ data \ file or relative to the current directory using. \ File. Any other name will be interpreted as a file naming described in THEOS.

For example, the following statement indicates a description of THEOS file and will cause the program to create the file \ txt \ bill.

OPEN # 1: "factura.txt" OUTPUT SEQUENTIAL

. If the file you want to open is really factura.txt and refers to the current directory, the sentence should have been:

OPEN # 1 ". \ Factura.txt" OUTPUT SEQUENTIAL

14.5 The default search sequence

Application Builder Builder W/32Application behaves like Windows programs Cundo seek an executable. The search sequence is:

- The directory where the calling program is executed.
- The current working directory
- The \ Windows or \ Windows \ system
- The directories set in the PATH
- Directories within the variable set B_LKPATH

14.6 The number of open files

By default, an application can open up to 64 files at once. Forty-eight of these can be opened as indexed (ISAM). Note, that the more open files, your system will need more memory. See chapter Compiler w/32, if you have questions about memory requirements.

14.7 CSI/CSH sentences

The sentences CSI / CSH should run only in 16-bit programs. These statements should be replaced with a call to cWinExec or cCreateProcess when developing 32-bit programs.

We strongly environment BASIC leave to perform other operations, eliminating the judgments CSI / CSH whenever possible. Have developed spatial files related functions for this purpose. (Covered in Chapter routines built in C language).

The sentences CSI / CSH write the command to execute a batch file in the directory \ tmp. Given that generates a batch file can enter multiple commands separated by <Intro/Nueva <LINE generated by CHR $ (13) + CHR (10).

For example, to change to the \ tst tst.exe delete the file and enter:

CMD $ = "CD \ TMP" + CHR $ (13) + CHR $ (10) + "THE TST.EXE" CSI CMD $

Your program will wait until you run the last command that was passed to the judgment CSI. While running a command CSI, an icon appears on-screen MS-DOS. When the command finishes CSI, the icon disappears and your program is still running.

Because some commands that are passed to CSI require keyboard input, we have added an option to the judgment CSI. If the first seven characters are past "noicon:" then the MS-DOS window appears as another window (inside the screen) allowing the user to accept data. Your program will be suspended until the command finishes.

For example, to format a floppy disk in drive A:

CMD $ = "noicon: FORMAT A:"

CSI CMD $

In this case, an MS-DOS window appears on your screen with the FORMAT command message: "Insert a new diskette in drive A: and press ENTER when ready". The user should press [Enter] and wait while the disk is formatted. When the format command completes, the MS-DOS window disappears and the program will continue running.

Also note that if some of the commands passed to CSI has some output that should be read by another command (pipe, redirection), need to use the option noicon since you may need to enter a PAUSE MS-DOS (Command Internal PAUSE) to finish running.

For example, if you use the DIR command in a CSI, by default, the window disappears as soon as the command finishes executing. Most likely not have time to read the command output. The following command would solve this problem:

CMD $ = "noicon: DIR + CHR $ (13) + CHR $ (10) +" PAUSE "

CSI CMD $

14.8 Msgbox Sentence

The message boxes are often used in Windows programs. The message "Application Exit" displayed from the File menu when exit is a typical example. The message is displayed until the operator "click" OK.
Posts boxes can be used from anywhere in your program as you use a PRINT statement. The syntax is:
MSGBOX "cadena_dada"
For example:
MSGBOX "Is the paper loaded in the printer?"
cMessageBoxOkC is a similar function that displays a box of OK / Cancel the user to continue or cancel the operation.

14.9 Printing

14.9.1 Native printing

When a program opens the file w/32 "PRINTERx" where x is from 1 to 9, it automatically creates a spooler file in the \ tmp. When the channel or associated with the file descriptor closes this printing is sent to the printer.
The first phase for printing "real" Windows has already been implemented for you to select a specific printer, or allow the user to select one via the print dialog. In any case, printing is still ASCII characters (with control sequences "embedded"). The next phase will allow you to send an image to the printer.
To use the new printer interface, check the following environment variable in the initialization file W32APP.w32, default:
B_WINPRINT = 1
Note that if B_WINPRINT not activated, or has a value other than one Windows application used printing method. "Bat" documented at the end of this section.
When used B_WINPRINT, B_PRINTERn variable can also be used for specifying the printer on one of the following ways:
• If B_PRINTERn not activated or has the value DEFAULT eg "B_PRINTER1 = DEFAULT", then it will use the system default printer specified in the Printer Manager. As soon as you close the channel, work automatically switches to Print Manager.
• They define a specific printer setting "B_PRINTERn = printer name". The Printer Manager will print control in the same way it does when the variable is set to DEFAULT.
Under Windows NT and 95, this has to be the name you assign to the printer in the Printer Manager. Under Windows (3.1,3.11), there is no option to put names to the printers, so use the name of the driver that is connected (for example, "HP LaserJet III").

Note that this is the only case where the number of PRINTER, 1-9, in the name of the variable is associated with the LPT port for that printer. If the variable has the value DEFAULT B_PRINTERn or DIALOG, there is no association between the number of printer and LPT port under any environment.
The function can be used to obtain cEnumPrinters printers information available. Typically, this feature is used from a BASIC program when you need information from printers to specify the variable name B_PRINTERn to particular through Bputenv function call. Refer to the job description for an example w/32.
• If the variable equals B_PRINTERn DIALOG, then the user will see a print dialog, allowing you to select a printer from the list specified in the Print Manager.
Since you will be printing ASCII text yet, there will be some options in the print dialog as "Copies" and selecting specific pages "Page Range" that have been disabled. This does not take effect until phase 2 is implemented.
Also note that if you change the default printer in the "Print Setup" from the print dialog, the selected printer get displayed only during the current program. The change has no effect on a program that has been made or CHAIN LINK. The "Network" is available only for you to connect another printer on the network, the rule above that will also be in effect.
B_WINPRINT If the variable is not set, the print mechanism for a file depends on the following:
• What is the operating system you are working with?.
• Does the printer is local, that is connected to your computer or a remote printer (local network)?
• Does the Remote Printer works under Windows for Workgroups, or some other type of LAN?
Because there are many variables involved in the printing process, w/32 runs a batch file (. BAT) to send the file to the printer. The batch file is called CETPRNTX.BAT (default will \ cetbin) where 'x' corresponds to the number of printer. For example PRINTER and execute both cetprnt1.bat PRINTER1, PRINTER2 cetprnt2.bat executed, and so on.

The cetprnt1.bat file included with the compiler contains only comments and not print any file. To enable printing must enter the appropriate command within the environment in which it is. The comments in this file contains two examples to run a Windows and other Windows NT.
To determine which command should be used first make sure you have a printer and is correctly installed. Then determine the required command from an MS-DOS window. For example if you are running Windows cetprnt1.bat send the file to the printer by entering the command:
COPY CETPRNT1.BAT PRN
Note that the DOS PRINT command can not be used from Windows as a resident, but can be used under Windows NT.
Once you can print from MS-DOS, cetprnt1.bat edit the file and insert the commands you want. A single line for Windows is:
REM COPY% 1 PRN
Delete All Entries in the file containing the following line:
COPY% 1 PRN
The name of the file to be printed is passed as a parameter to the file. Bat (% 1 will be replaced by the filename). After modifying the file cetprnt1.bat, check printing process any text file as config.sys entering:
CETPRNT1 \ CONFIG.SYS
The last step you must take is to generate a file. Bat for each printer that has.

14.9.2 Printing text in Windows printers not compatible with MS-Dos

There are some printers are not compatible with MS-DOS, to solve this has created a tool that converts plain text into text graphic (w32prt.exe). To make this program functional must have the following variables The Area of CET established as follows:
• Variables in w32app.w32:
B_WINPRINT = 0
B_PRINTERORIENT = H
* Orientation V, H, VIRTUAL
B_PRINTFSIZE = 10
* Font Size
And there must be w32prt.exe, w32prtl.exe, CETPRNT1.BAT CETBIN directory.

14.9.3 GRAPHICS PRINTING WITH WINDOWS SOURCES

14.9.3.1 CBEPrint.dll class file

classes
• Printer
The Printer class provides simplified access to the Windows print spooler 95 / NT

14.9.3.2 Class Printer

Buffer line and word-level justification
The print object remains in the line buffer all text sent by Text or calls TextAt. The line buffer is printed when a call to NewLine, or when it reaches \ n in the input text.
If the length of the text to be printed is greater than the width of the page, the print object may discard the rest of the line or continue it in the beginning of the next line. In the latter case the justification is performed at word. The justification mode in case the line buffer length is greater than the width of the page is determined by the call to WordWrap. The default value for the property WordWrap is 0 for absolute mode and one for virtual mode.
Modes
There are two print modes-absolutely and virtual mode. The mode is set by calling Create and can not be changed for printing an object already created.
In virtual mode printing some calls working with coordinates relative to the current virtual page. The coordinates on the page can be set by the call to setPage.
The following calls have special behavior in virtual mode:
1. Text, TextAt, newPosition, Position working with coordinates relative to the current virtual page.
2. The default value for the property WordWrap is 1.
3. The print object does not advance to a new physical page automatically. Instead, when the printing exceeds the current virtual page, the print object maintains the rest of the line in the line buffer (until the first "\ n"). The application program must checkear line buffer after each call to NewLine or Text to determine if the text

contains a "\ n". This check can be done with the call IsEmpty. If the line buffer is not empty, this means that the virtual page is complete. The application program in the same physical page, or by calling NewPage, then set the virtual page at the following physical and NewLine called again to print the rest of the line remains in the line buffer.
4. The print object drops all input content after the first sequence "\ n". This restriction will be removed in a future release.

Methods

Create (name, mode) creates the object, not the print job begins.

Name (String) - The name of the printer, open the selection box if empty.
Mode (String) - print lock modes:
"" - Normal mode (virtual mode disabled and need guidance to default printer)
"Virtual" - Virtual Mode
"Lanscape" - landscape orientation
"Portrait" - Flip

CALL:
```
CALL cetObjectCreate (ADDROF (obj%), "Printer", $ name, $ mode, 2)
```
MACRO:
```
PrinterCreate (obj%, $ name, $ mode)
```

CreateCommon (name, mode) creates a common object, not the print job begins

Name (String) - The name of the printer, open the selection box if empty.
Mode (String)-print modes opening

CALL:
CALL cetObjectCreateCommon (ADDROF (obj%), "Printer", $ name, $ mode, 2)
MACRO:
PrinterCreateCommon (obj%, $ name, $ mode)

Destroy () destroys the object, cancel a print job is not yet finished

CALL:
CALL cetObjectDestroy (obj%)
MACRO:
PrinterDestroy (obj%)

Begin (name) Starts a docuemnto. Each document is a separate print job in the spooler. The print spooler ensures that the document will not mix with others.

Name (String) - name of the document. The document name appears in the print status.

CALL:
CALL cetObjectExecute (obj%, "Begin", $ name, 1)
MACRO call:
PrinterBegin (obj%, $ name)

End () End document

CALL:
CALL cetObjectExecute (obj%, "End", 0)
MACRO:
PrinterEnd (obj%)

Cancel () Cancels document

CALL:
CALL cetObjectExecute (obj%, "Cancel", 0)
MACRO:
PrinterCancel (obj%)

Font (NombreFuente, size, style) Sets the font for text output

NombreFuente (String) - The name of the source
Size (Large) - Font size
Style (String) - one or more of the styles: B [old], I [Talic], [U] nderline (Bold, Italic, Underline)

CALL:
CALL cetObjectExecute (obj%, "Font", NombreFuente $,% size, $ style, 3)
MACRO:
PrinterFont (obj% $ NombreFuente, size%, $ style)

FontColor (Number, R, V, A) activates the font color, the original color value is 0,0,0 (black)

Number (integer) - number of source
R (integer) - value of R (red) color; Valid values are 0 .. 255
V (integer) - value of component V (green) color; Valid values are 0 .. 255
A (integer) - value of component A (blue) color; Valid values are 0 .. 255

CALL call:
CALL cetObjectExecute (obj% "FontColor" num% r% v% at%, 4)
MACRO call:
PrinterFontColor (obj% num%, r%, v%, y%)

Text (String) prints a text string starting at the current cursor position.

String - The text string can contain the following escape characters:
\ \ - Backslash
\ N - newline
\ F - end of page
\ R - right alignment
\ L - left alignment
\ C - centered
\ J - justified
\ 0 .. 9 - Source Selection

Calll:
CALL cetObjectExecute (obj%, "Text", string, 1)
MACRO:
PrinterText (obj%, $ string)

TextAt (Position, String) prints a text string from the position indicated

Position - Position in pixels.
String - String that can contain the following escape characters:
\ \ - Backslash
\ N - newline
\ F - end of page
\ 0 .. 9 - Source Selection

CALL:
CALL cetObjectExecute (obj%, "Text", position%, $ string, 2)
MACRO:
PrinterTextAt (obj% position%, $ string)

NewLine () is positioned at the beginning of the next line

CALL:
CALL cetObjectExecute (obj%, "NewLine", 0)
MACRO:
PrinterNewLine (obj%)

NewLine (position) is positioned at the beginning of the line in the vertical displacement given

Position (Integer) - New vertical

CALL:
CALL cetObjectExecute (obj%, "NewLine" position% 1)
MACRO:
PrinterNewPosition (obj% position%)

NewPage () new page

CALL:
CALL cetObjectExecute (obj%, "NewPage", 0)
MACRO:
PrinterNewPage (obj%)

Left () aligns the text to the left

CALL:
CALL cetObjectExecute (obj%, "Left", 0)
MACRO:
PrinterLeft (obj%)

Right () aligns the text to the right

CALL:
CALL cetObjectExecute (obj%, "Right", 0)
MACRO:
PrinterRight (obj%)

Center () centers the text

CALL:
CALL cetObjectExecute (obj%, "Center", 0)
MACRO:
PrinterCenter (obj%)

Justify () justifies text

CALL:
CALL cetObjectExecute (obj%, "Justify", 0)
MACRO:
PrinterJustify (obj%)

SetPage (Left, Top, Width, Height) sets the position and size of the virtual page
If the virtual page specified does not match the physical page, or any of the coordinates is <0, then the virtual page is equal to the physical page

Left (Integer) - absolute horizontal coordinate of the upper left corner in pixels
Top (Integer) - absolute vertical coordinate of the upper left pixel
Width (Integer) - virtual page width in pixels
Height (Integer) - Height of the virtual page in pixels

CALL:
CALL cetObjectExecute (obj%, "VirtPage" Left%, top%, width%, height%, 4)
MACRO:
PrinterSetPage (obj%, Left%, top%, width%, height%)
PrinterResetPage (obj%)

DrawLine (DesdeX, DesdeY, HastaX, HastaY, width, style) Draws a line between two points

DesdeX (Integer) - Horizontal position absolute starting point
DesdeY (Integer) - absolute vertical position of the starting point
ToX (Integer) - Horizontal position absolute destination point
Toy (Integer) - Vertical position absolute destination point
Width (Integer) - Line width in pixels
Style (Integer) - Line Style:
0 - solid line
1 - Stripes
2 - Points
3 - Raya-point
4 - dash-dot-dot

CALL:
CALL cetObjectExecute (obj%, "Line", DesdeX% DesdeY% HastaX% HastaY% Width% Style%, 6)
MACRO:
PrinterDrawLine (obj% DesdeX% DesdeY% HastaX% HastaY% Width% Style%)

DrawBox (Left, Top, Width, Height, AnchuraLinea, Style) Draw a rectangle

Left (Integer) - absolute horizontal coordinate of the upper left corner
Top (Integer) - absolute vertical coordinate of upper left corner
Width (Integer) - Width of rectangle
Heigth (Integer) - Height of the rectangle
AnchuraLinea (Integer) - Width, in pixels, of the rectangle boundary line
Style (Integer) - Line Style:
0 - solid line
1 - Stripes
2 - Points
3 - Raya-point
4 - dash-dot-dot

CALL:
CALL cetObjectExecute (obj%, "Box", Left%, top%, width%, height, AnchuraLinea% Style%, 6)
MACRO:
PrinterDrawBox (obj%, Left%, top%, width%, height, AnchuraLinea% Style%)

DrawBitmap (Name, Left, Top, Width, Height) Prints a bitmap stored in a file whose name is specified. The size fits the Width and Height.

Name (String) - File name and path (Nomenclature WINDOWS, not THEOS)
Left (Integer) - absolute horizontal coordinate of the upper left corner
Top (Integer) - absolute vertical coordinate of upper left corner
Width (Integer) - Width of the bitmap
Height (Integer) - Height of the bitmap

CALL:
CALL cetObjectExecute (obj%, "Bitmap", $ name, Left%, top%, width%, height%, 5)
MACRO:
PrinterDrawBitmap (obj%, $ name, Left%, top%, width%, height%)

Properties

Status (Integer, State) returns the current state of the print
0 - Ok,
> 0 - Error

CALL:
CALL cetObjectGetProperty (obj%, "Status", ADDROF (status%))
MACRO:
PrinterStatus (obj%, status%)

PageWidth (Integer, width) Returns the page width in pixels

CALL:
CALL cetObjectGetProperty (obj%, "PageWidth" ADDROF (W%))
MACRO call:
PrinterPageWidth (obj% Width%)

PageLength (Integer, Long) Returns the length of the page in pixels

CALL:
CALL cetObjectGetProperty (obj%, "PageLength" ADDROF (length%))
MACRO:
PrinterPageLength (obj%, length%)

PageSizeY (Integer, Long) Returns the page length in millimeters

CALL call:
CALL cetObjectGetProperty (obj%, "PageSizeY" ADDROF (length%))
MACRO call:
PrinterPageLength (obj%, and%)

PageSizeX (Integer, width) Returns the page width in millimeters

CALL call:
CALL cetObjectGetProperty (obj% "PageSizeX" ADDROF (w%))
MACRO call:
PrinterPageLength (obj% x%)

Position (Integer, Position) Actual relative vertical position (virtual page) in pixels

CALL:
CALL cetObjectGetProperty (obj%, "Position", ADDROF (position%))
MACRO:
PrinterPosition (obj% position%)

AbsPosition (Integer, Position) Actual absolute vertical position in pixels

CALL:
CALL cetObjectGetProperty (obj%, "AbsPos" ADDROF (position%))
MACRO:
PrinterAbsPosition (obj% position%)

IsEmpty (Full, Empty) Status line buffer, 1 - empty, 0 - not empty

CALL:
CALL cetObjectGetProperty (obj%, "Empty", ADDROF (empty%))
MACRO:
PrinterIsEmpty (obj% empty%)

WordWrap (Integer, JP) Enables justification ownership level words, if (1), No (0)

CALL:
CALL cetObjectSetProperty (obj%, "WordWrap" JP%)
MACRO:
PrinterWordWrap (obj%, JP%)

Notes
1. The sources are numbered starting from 1 in the order they are created by calling Font. The font number 0 is the default print font printer specified when making the call to Create.
We recommend creating all sources that are required immediately after the call to Create, and then select the fonts by number.
2. The size and position of the pages in the call to newPosition are expressed in pixels.